# DEVELOPMENT AND EVALUATION OF METHODOLOGIES FOR MAINTAINING SOFTWARE CONTAINERS

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

**DOCTOR OF PHILOSOPHY**
by

**ANJALI BANSAL**
Roll No.2K21/PHDSE/01

Under the Supervision of
**PROF. RUCHIKA MALHOTRA**
Professor and Head of Department,
Department of Software Engineering
and
**DR. MAROUANE KESSENTINI**
Dean, College of Computing,
Grand Valley State University, Michigan, USA




**Department of Software Engineering**

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

**October, 2026**

Copyright ©October, 2026
Delhi Technological University, Shahbad Daulatpur,
Main Bawana Road, Delhi 110042
All rights reserved



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

# CANDIDATE'S DECLARATION

I, **Anjali Bansal (2K21/PHDSE/01)**, hereby declare that the work which is being presented in the thesis entitled **"Development and Evaluation of Methodologies for Maintaining Software Containers"** 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 **2021** to **2026** under the supervision of **Prof. Ruchika Malhotra** and **Dr. Marouane Kessentini**.

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.

**Anjali Bansal**

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.

*kessentini*

**Signature of Supervisor** **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 **Anjali Bansal (2K21/PHDSE/01)** has carried out her research work presented in this thesis entitled **"Development and Evaluation of Methodologies for Maintaining Software Containers"** for the award of **Doctor of Philosophy** from the Department of Software Engineering, Delhi Technological University, Delhi, under my supervision. The thesis embodies results of original work, and studies are carried out by the student himself, and the contents of the thesis do not form the basis of 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

**Dr. MAROUANE KESSENTINI**

Dean, College of Computing

Grand Valley State University

Michigan, USA

kessentini

Date:

# Acknowledgment

First and foremost, I express my heartfelt gratitude to the Almighty for His endless blessings, strength, and guidance throughout my academic journey. His grace has been a constant source of inspiration and resilience during both challenging and rewarding moments.

I would like to express my deepest appreciation to my supervisor, **Prof. Ruchika Malhotra**, Professor and Head of the Department of Software Engineering at Delhi Technological University, for her invaluable guidance, encouragement, and unwavering support. Her insightful feedback, thoughtful suggestions, and readiness to engage in meaningful discussions helped me navigate various challenges. I am sincerely grateful for her patience, dedication, and mentorship, which have significantly contributed to both my academic and personal growth. This research would not have been possible without her continued support and confidence in my abilities.

I am equally grateful to my co-supervisor, **Dr. Marouane Kessentini**, Dean of the College of Computing at Grand Valley State University, for his expertise, constructive suggestions, and thoughtful guidance. His perspectives and critical insights significantly enriched the quality and depth of this work.

I would also like to express my sincere gratitude to Delhi Technological University for providing me with the opportunity and an enriching academic environment to carry out this research.

I extend my deepest love and gratitude to my father, **Mr. Jai Prakash Bansal** and my mother, **Mrs. Rekha Bansal**, whose unwavering support, sacrifices, and encouragement have been the foundation of my achievements. Their belief in me has been my greatest strength and inspiration throughout this journey.

Finally, I would like to thank my **friends** for their companionship, encouragement, and positivity, which brought balance and joy during this demanding journey. I am truly grateful to all who supported me, directly or indirectly, in achiev-

ing this milestone.

Thank you all for being a part of this journey.

**Anjali Bansal**

# Abstract

New technologies and innovations are being introduced day by day. This progress results in the improvement of the existing technologies. Nowadays, cloud computing is gaining tremendous attention to deliver information via the internet. Virtualization plays a major role in cloud computing as it deploys multiple virtual machines on the same physical machine and thus results in improving resource utilization. Hypervisor-based virtualization and containerization are two commonly used approaches in operating system virtualization. Container-based virtualization has emerged as a transformative technology in modern cloud computing, offering lightweight, portable, and efficient deployment solutions compared to traditional hypervisor-based virtualization. Containers are standard, secure, and lightweight virtualization technologies. The use of container technology has been increased during the last few years, with Docker as the leading container platform. Docker's online repository for publicly available container images, called Docker Hub, hosts over 3.5 million images. Docker hub is the world's largest repository which contains docker images. As containerization adoption accelerates, particularly through platforms like Docker, critical challenges persist in ensuring secure, efficient, and reliable container maintenance throughout their lifecycle.

Software containerization has revolutionized the deployment and maintenance of modern applications by providing lightweight, portable, and scalable environments. With Docker and related technologies becoming central to cloud-native development, the need for effective methodologies to ensure container reliability, performance, and security has become increasingly critical. This research aims to

develop, evaluate, and integrate intelligent, secure, and efficient frameworks for the maintenance of software containers across their lifecycle. This research addresses these challenges through an integrated framework combining systematic literature review, federated learning, deep learning, hyperparameter optimization, and comprehensive security and performance evaluation.

The research is systematically organized into several phases, each concentrating on different aspects on maintenance of software containers in cloud computing environment. The first phase involves a comprehensive systematic literature review to identify the techniques used for detecting docker image to build the container, various tools and techniques used to find the security vulnerabilities and resource utilization, and various approaches used in scheduling the container on working nodes. This review provides a foundation for understanding the current state of research and identifying gaps that this work aims to address. This review systematically investigates four critical phases of software container maintenance: container image detection, container scheduling, security vulnerability analysis, and performance evaluation. A comprehensive systematic literature review of 145 primary studies (2017-2022) reveals fragmented research focus, with significant gaps in intelligent scheduling mechanisms, proactive vulnerability auditing, and deployment-level performance monitoring. Building upon these insights, this work proposes novel methodologies integrating machine learning-driven predictive models with containerized deployment strategies.

The thesis presents multiple innovative contributions. First, a federated learning-based framework for cloud workload forecasting demonstrates superior privacy-preserving prediction capabilities across distributed environments using real-world datasets (Alibaba Cluster Trace and Google Cluster Trace). Second, optimized deep learning models incorporating OPTUNA-based hyperparameter tuning and optimized federated learning models incorporating meta heuristic algorithm such as Grey Wolf Optimization (GWO) and whale optimization (WOA), achieve enhanced accuracy in cloud failure prediction (job and task failures) and software

defect prediction. Third, a stacking ensemble deep learning model (DeepHAR) integrating CNN, LSTM, BiLSTM, and hybrid architectures attains state-of-the-art performance in human activity recognition.

Critically, all proposed models are deployed within Docker containers and subjected to rigorous security vulnerability scanning using multiple open-source tools (Anchore, Aqua Trivy, Snyk, Clair) and comprehensive resource utilization monitoring using benchmark tools and real-time monitoring frameworks (cAdvisor, Prometheus, Grafana). Experimental results reveal that while containerized models demonstrate computational efficiency and scalability, persistent medium-to-high severity vulnerabilities necessitate continuous security auditing in production environments.

Experimental validation demonstrates that the proposed models outperform traditional centralized and untuned baselines in terms of predictive accuracy, security resilience, and resource efficiency. This research advances the state-of-the-art by bridging the gap between predictive model innovation and secure, resource-efficient deployment. The proposed end-to-end framework provides actionable insights for practitioners developing trustworthy, privacy-preserving, and operationally resilient AI-driven systems in containerized cloud environments. By integrating predictive modeling, optimization, and secure container deployment, this work provides a unified framework that advances the reliability and maintainability of software containers in large-scale cloud computing environments. The outcomes contribute toward the development of intelligent, privacy-preserving, and trustworthy AI-driven solutions in software engineering and cloud infrastructure management. By unifying predictive modeling, federated optimization, and secure container deployment, this thesis lays a foundational framework for next-generation, privacy-preserving, and sustainable cloud intelligence systems

# Contents

**List of Tables** **xvi**

**List of Figures** **xvi**

**List of Publications** **xvi**

**Abbreviations** **xviii**

**1 INTRODUCTION** **1**

1.1 Introduction . . . . . 1

1.1.1 What is Container? . . . . . 2

1.1.2 What is Docker, Docker image, and Dockerfile? . . . . . 3

1.2 Containerization Technologies . . . . . 3

1.3 Container Orchestration Tools . . . . . 4

1.4 Phases in Maintenance of Software Containers . . . . . 6

1.4.1 Container Image Detection . . . . . 6

1.4.2 Container Scheduling . . . . . 6

1.4.3 Security Analysis of Containers . . . . . 8

1.4.4 Container Performance Evaluation Analysis . . . . . 8

1.5 End-to-End Predictive Modeling and Deployment . . . . . 9

1.5.1 Key Phases of the Process . . . . . 9

1.6 Literature Survey . . . . . 11

1.7 Motivation and Objective of the Thesis . . . . . 19

1.7.1 Vision . . . . . 20

1.7.2 Focus . . . . . 20

1.7.3 Goals . . . . . 21

1.8 Overview of the Work . . . . . 25

1.9 Organization of the Thesis . . . . . 28

**2 RESEARCH METHODOLOGY 32**

2.1 Introduction . . . . . 32

2.2 Research Process . . . . . 33

2.3 Research Problem Identification . . . . . 34

2.4 Literature Survey . . . . . 35

2.5 Defining Study Variables . . . . . 36

2.5.1 Independent Variables . . . . . 36

2.5.2 Dependent Variables . . . . . 40

2.6 **Data Analysis Techniques** . . . . . 42

2.6.1 Decision Tree . . . . . 44

2.6.2 K Nearest Neighbor . . . . . 44

2.6.3 Logistic Regression . . . . . 45

2.6.4 Naïve Bayes . . . . . 45

2.6.5 Support Vector Machine . . . . . 46

2.6.6 Random Forest . . . . . 47

2.6.7 Stacking . . . . . 48
2.6.8 Artificial Neural Network (ANN) . . . . . 48
2.6.9 Convolutional Neural Network (CNN) . . . . . 49
2.6.10 Long Short-Term Memory (LSTM) . . . . . 50
2.6.11 Bi-Directional LSTM (BiLSTM) . . . . . 51
2.6.12 CNN-LSTM . . . . . 52
2.6.13 CNNBiLSTM . . . . . 52
2.6.14 Federated Learning (FL) . . . . . 53
2.7 **Empirical Data Collection** . . . . . 54
2.8 Data Preprocessing . . . . . 56
2.9 Data Imbalance Handling . . . . . 58
2.10 Development and Validation of Predictive Models . . . . . 58
2.11 Performance Evaluation Metrics . . . . . 59
2.12 Statistical Test . . . . . 64
2.13 **Model Deployment** . . . . . 66
2.14 **Security Vulnerability Analysis** . . . . . 66
2.15 **Resource Consumption Analysis** . . . . . 68

**3 SYSTEMATIC LITERATURE REVIEW 70**

3.1 Introduction . . . . . 70
3.1.1 Containerization Technologies . . . . . 71
3.1.2 Container Orchestration Tools . . . . . 72
3.2 Research Methodology . . . . . 74
3.2.1 Research Questions . . . . . 75

3.2.2 Search Strategy . . . . . 76
3.2.3 Inclusion and Exclusion criteria . . . . . 77
3.2.4 Quality Assessment Questions . . . . . 78
3.3 Review Results . . . . . 78
3.3.1 RQ 1 Results . . . . . 79
3.3.2 RQ 2 Results . . . . . 81
3.3.3 RQ 3 Results . . . . . 85
3.3.4 RQ 4 Results . . . . . 99
3.3.5 RQ 5 Results . . . . . 103
3.4 Conclusion and Future Guidelines . . . . . 117

**4 WORKLOAD FORECASTING 120**

4.1 Introduction . . . . . 120
4.2 Research Methodology . . . . . 122
4.2.1 Dataset . . . . . 122
4.2.2 Data Preprocessing . . . . . 123
4.2.3 Techniques . . . . . 124
4.2.4 Performance Analysis . . . . . 125
4.3 Experimental Result . . . . . 126
4.4 Conclusion . . . . . 128

**5 DEVELOPMENT OF CLOUD FAILURE PREDICTION MODELS 130**

5.1 Introduction . . . . . 130
5.2 Research Methodology . . . . . 133
5.2.1 Dataset . . . . . 133

5.2.2 Data Preprocessing . . . . . 134

5.2.3 Techniques . . . . . 137

5.3 CFP with Optimized Ensemble of Deep Learning Architectures . . 138

5.3.1 Experimental Framework . . . . . 139

5.3.2 Result Analysis . . . . . 140

5.3.3 **Comparison with Existing Studies** . . . . . 146

5.4 CFP with Federated Learning Architecture . . . . . 148

5.4.1 Experimental Framework . . . . . 148

5.4.2 Result Analysis . . . . . 149

5.5 Conclusion . . . . . 150

**6 SECURITY AND PERFORMANCE ANALYSIS OF DOCKERIZED SOFTWARE DEFECT PREDICTION MODEL 151**

6.1 Introduction . . . . . 151

6.2 Research Methodology . . . . . 153

6.2.1 Dataset . . . . . 154

6.2.2 Data Preprocessing . . . . . 154

6.2.3 Techniques . . . . . 155

6.2.4 Performance Evaluation Measure . . . . . 159

6.2.5 Tools for Security Vulnerability Analysis . . . . . 159

6.2.6 Tools for Resource Consumption Analysis . . . . . 160

6.3 Security Analysis of Official and verified docker Images . . . . . 162

6.3.1 Experimental Framework . . . . . 163

6.3.2 Result Analysis . . . . . 164

6.4 SDP with Optimization of DL Techniques using Hybrid Meta-heuristic Approach . . . . . 168

6.4.1 Experimental Framework . . . . . 169

6.4.2 Result Analysis . . . . . 170

6.5 Security and Performance Analysis of Dockerized FL Model for SDP . . . . . 174

6.5.1 Experimental Framework . . . . . 175

6.5.2 Result Analysis . . . . . 177

6.6 Security Analysis of Optimized FL Model for SDP . . . . . 186

6.6.1 Experimental Framework . . . . . 187

6.6.2 Result Analysis . . . . . 189

6.7 Comparison with Baseline Techniques . . . . . 194

6.8 Conclusion . . . . . 196

**7 COMPREHENSIVE PERFORMANCE AND SECURITY ANALYSIS OF A DOCKERIZED ENSEMBLE DEEP LEARNING MODEL FOR HUMAN ACTIVITY RECOGNITION 198**

7.1 Introduction . . . . . 198

7.2 Research Methodology . . . . . 200

7.2.1 Dataset . . . . . 201

7.2.2 Techniques . . . . . 202

7.2.3 Performance Evaluation Measure . . . . . 206

7.2.4 Tools for Security Vulnerability Analysis . . . . . 207

7.2.5 Tools for Resource Consumption Analysis . . . . . 207

7.2.6 Hardware Configuration Details . . . . . 208

7.3 Experimental Results . . . . . 208
7.3.1 Analysis on WISDM Dataset . . . . . 209
7.3.2 Analysis on UCI HAR Dataset . . . . . 210
7.3.3 Analysis on FLAAP Dataset . . . . . 211
7.3.4 Statistical Test . . . . . 212
7.3.5 Security Analysis . . . . . 214
7.3.6 Performance Analysis . . . . . 216
7.4 Conclusion . . . . . 219

**8 CONCLUSION** **221**

8.1 Summary of the Research Work . . . . . 221
8.2 Application of the Work . . . . . 222
8.2.1 Aid to Researchers . . . . . 222
8.2.2 Aid to Software Practitioners . . . . . 223
8.2.3 Aid to industry . . . . . 224
8.2.4 Aid to Society . . . . . 225
8.3 Future of the Work . . . . . 225

**Supervisor's Biography** **275**

# List of Tables


2.1 Metrics Suite . . . . . 37

2.2 Event Types . . . . . 41

2.3 Confusion Matrix Representation . . . . . 60

3.1 Containerization Technologies . . . . . 71

3.2 Comparison of Docker Swarm, Kubernetes, and Mesos . . . . . 73

3.3 Research Questions . . . . . 75

3.4 Quality Assessment Criteria . . . . . 79

3.5 Classification of Studies . . . . . 79

3.6 Dataset Used . . . . . 85

3.7 Performance Objectives Used in Container Scheduling . . . . . 86

3.8 Summary of Studies Related to Heuristic Approaches for Container Scheduling . . . . . 89

3.9 Findings Related to Meta Heuristic Approaches for Container Scheduling . . . . . 92

3.10 Findings Related to Mathematical Modeling Approaches for Container Scheduling . . . . . 95

3.11 Findings Related to ML Approaches for Container Scheduling . . 97

3.12 Summary of studies on container security approaches and findings 100

3.13 Findings Related to Approaches to Detect Security Attacks . . . . 102

3.14 Correlation between Technique and Performance Evaluation Metrics . . . . . 106

3.15 Correlation between Benchmark Tools and Evaluation Metrics . . 107

3.16 CPU Intensive Benchmarks . . . . . 108

3.17 Memory Intensive Benchmarks . . . . . 111

3.18 Disk I/O Intensive Benchmarks . . . . . 112

3.19 Network I/O Intensive Benchmarks . . . . . 114

3.20 Container Performance Monitoring Tools . . . . . 115

4.1 Statistical characteristics of dataset . . . . . 123

4.2 Hyperparameter Details . . . . . 125

4.3 Experimental Results . . . . . 126

5.1 Summary of Categories, Types, and Causes of Cloud Failures . . . 131

5.2 Independent and dependent features for task level and job level termination dataset . . . . . 136

5.3 Comparison between the performance of base models and proposed model for task level failure dataset . . . . . 141

5.4 Comparison between the performance of base models and proposed model for job level failure dataset . . . . . 142

5.5 Comparative Analysis . . . . . 146

5.6 Performance comparison between CNN and Federated Learning with CNN. . . . . 149

6.1 Statistical data about datasets . . . . . 154

6.2 Hyperparameter Values . . . . . 157

6.3 Hyperparameter details . . . . . 157

6.4 Hyperparameter values of the proposed model . . . . . 159

6.5 Container scanning tools used . . . . . 160

6.6 Official Images . . . . . 164

6.7 Total number of vulnerabilities reported by each tool . . . . . 165

6.8 Severity wise vulnerabilities reported in each image . . . . . 166

6.9 Table . . . . . 171

6.10 Mean ranks based on accuracy and AUC values . . . . . 174

6.11 AUC and MCC values for CNN and ANN . . . . . 178

6.12 Severity wise total number of vulnerabilities identified . . . . . 178

6.13 Identified security vulnerabilities . . . . . 179

6.14 Number of users and spawn rate . . . . . 179

6.15 Resource usage using monitoring tools . . . . . 181

6.16 Performance comparison of FLCNN with HPT and FLANN with HPT . . . . . 190

6.17 Severity wise security vulnerabilities for each tool . . . . . 192

6.18 Threats with the severity level . . . . . 193

6.19 Comparative analysis of proposed model with conventional approaches . . . . . 194

6.20 Mean rank of all models . . . . . 195

7.1 Summarization of Characteristics of Datasets . . . . . 201

7.2 Hyperparameter Details . . . . . 204

7.3 AUC and MCC values for WISDM, UCI-HAR, and FLAAP . . . 209

7.4 Mean Ranks based on AUC values . . . . . 213

7.5 Mean Ranks based on MCC Values . . . . . 213

7.6 Security Vulnerabilities . . . . . 215

7.7 CVEs Identified . . . . . 215

7.8 Resource Consumption using Container Monitoring Tools . . . . 217

# List of Figures

1.1 Architectural comparison between virtual machine and container . 2

1.2 End to End Predictive Modeling and Deployment . . . . . 10

2.1 Research Methodology . . . . . 34

2.2 Traditional vs. FL . . . . . 53

3.1 Categorization based on Publication Type and Publication Year . 80

3.2 Distribution of Primary Studies by Publisher Over Years . . . . . 80

3.3 Primary Studies Selected in Each Phase . . . . . 81

3.4 Dataset Used . . . . . 85

3.5 Scheduling Techniques . . . . . 89

3.6 Performance Objectives Used in Heuristic Approaches . . . . . 92

3.7 Performance Objectives Used in Meta Heuristic Approaches . . . 94

3.8 Performance Objectives Used in Mathematical Modeling Approaches 96

3.9 Performance Objectives Used in ML Approaches . . . . . 98

3.10 Benchmark Tools Used . . . . . 107

3.11 CPU Intensive Benchmark Tools Used . . . . . 110

3.12 Memory Intensive Benchmark Tools Used . . . . . 111

3.13 Disk I/O Intensive Benchmark Tools Used . . . . . 113

3.14 Network I/O Intensive benchmark tools . . . . . 114

4.1 Research Framework . . . . . 124

4.2 Comparison between actual CPU usage rates and predicted CPU usage rates of the proposed model for alibaba cluster trace v2018, alibaba cluster trace 2020, GCT 2011, and GCT 2019 datasets. . . 127

4.3 Comparison of proposed approach with CNN and LSTM . . . . . 128

5.1 Fault Tolerance Approaches . . . . . 132

5.2 Correlation matrix of instances of dataset A . . . . . 135

5.3 Correlation matrix of instances of dataset B . . . . . 136

5.4 Proposed Framework . . . . . 140

5.5 Comparison of results of base learning models, random forest, and proposed model with and without HPT for task level termination 143

5.6 Comparison of results of base learning models, random forest, and proposed model with and without HPT for job level termination dataset . . . . . 145

5.7 Computational Efficiency Comparison of Models (Task level vs Job Level) . . . . . 146

5.8 Proposed Methodology . . . . . 149

6.1 Before oversampling and after oversampling . . . . . 155

6.2 Total number of vulnerabilities reported by each tool . . . . . 166

6.3 Severity wise vulnerabilities reported in each image . . . . . 167

6.4 CVEs Published per Year . . . . . 168

6.5 Proposed Architecture . . . . . 170

6.6 Comparison of proposed model with CNN, CNN_GWO, and CNN_WOA models . . . 173

6.7 Proposed Architecture . . . 176

6.8 Load testing using locust . . . 180

6.9 Prometheus CPU usage . . . 182

6.10 Prometheus memory usage . . . 182

6.11 Prometheus network traffic . . . 183

6.12 Prometheus total bytes used . . . 183

6.13 CPU usage using Grafana dashboard . . . 184

6.14 Memory usage using Grafana dashboard . . . 184

6.15 Network traffic using Grafana dashboard . . . 184

6.16 Proposed Architecture . . . 188

6.17 Comparative Analysis of FLCNN with HPT and without HPT in terms of AUC (a) and MCC (b) . . . 191

6.18 Heatmap for nemenyi post hoc test based on AUC score . . . 196

7.1 Architectural View of Our Proposed DeepHAR Model . . . 205

7.2 Activities in WISDM . . . 205

7.3 Activities in UCI HAR . . . 205

7.4 Activities in FLAAP . . . 206

7.5 Docker Container Monitoring Architecture . . . 208

7.6 Activity wise AUC and MCC values for WISDM dataset . . . 210

7.7 Activity wise AUC and MCC values for UCI HAR dataset . . . 211

7.8 Activity wise AUC and MCC values for FLAAP dataset . . . 212

7.9 Heat map of pairwise nemenyi post hoc scores based on AUC and MCC values . . . . . 214

7.10 Prometheus CPU usage . . . . . 217

7.11 Prometheus memory usage . . . . . 218

7.12 Prometheus network traffic received . . . . . 218

7.13 Grafana CPU usage . . . . . 219

7.14 Grafana memory usage . . . . . 219

# List of Publications

**Papers Accepted/Published in International Journals**

1. Ruchika Malhotra, Anjali Bansal, and Maroaune Kessentini, "A Systematic Literature Review on Maintenance of Software Containers", *ACM Computing Surveys*, 2024.

2. Ruchika Malhotra, Anjali Bansal, and Marouane Kessentini, "Deployment and Performance Monitoring of Docker Based Federated Learning Framework for Software Defect Prediction", *Cluster Computing*, 2024.

3. Ruchika Malhotra, Anjali Bansal, and Marouane Kessentini, "Assessing Security Vulnerabilities in a Docker-Enabled Federated Learning Framework with Hyperparameter Tuning for Software Bug Prediction", *International Journal of Information Security*, 2026.

**Papers Accepted/Published in International Conferences**

1. Ruchika Malhotra, Anjali Bansal, and Marouane Kessentini, "Vulnerability Analysis of Docker Hub Official and Verified Images", *In Proceedings of 17th IEEE International Conference on Service Oriented System Engineering (IEEE SOSE 2023)*, Athens, Greece, 2023.

2. Ruchika Malhotra, Anjali Bansal, and Marouane Kessentini, ”Towards Federated Learning Approach for Workload Prediction in Cloud Computing”, *In Proceedings of 5th International Conference on Data Analytics and Management (ICDAM 2024)*, 2024.

3. Ruchika Malhotra, Anjali Bansal, and Marouane Kessentini, ”A Hybrid Metaheuristic Approach: Grey Wolf and Whale Optimization for Deep Learning in Software Bug Prediction”, *In Proceedings of 6th International Conference on Data Analytics and Management (ICDAM 2025)*, 2025.

**Papers Communicated in International Journals**

1. Ruchika Malhotra, Anjali Bansal, and Marouane Kessentini, ”Performance Monitoring and Security Vulnerability Analysis of Docker Based Ensemble Learning Model for Human Activity Recognition”, *Engineering Applications of Artificial Intelligence*

2. Ruchika Malhotra, Anjali Bansal, and Marouane Kessentini, ”Intelligent Cloud Failure Prediction using Optimized Ensemble Models: A Data Driven Approach”, *Automated Software Engineering*

# Abbreviations

| | |
|---|---|
| **VM** | **Virtual Machine** |
| **OS** | **Operating System** |
| **LXC** | **Linux Containers** |
| **OpenVZ** | **Open Virtuozzo** |
| **HPC** | **High Performance Computing** |
| **AUC** | **Area Under the Curve** |
| **MCC** | **Matthews Correlation Coefficient** |
| **CC** | **Cloud Computing** |
| **ML** | **Machine Learning** |
| **DL** | **Deep Learning** |
| **LSTM** | **Long Short-Term Memory** |
| **BiLSTM** | **Bi-Directional Long Short-Term Memory** |
| **GRU** | **Gated Recurrent Unit** |
| **CNN** | **Convolutional Neural Network** |
| **DNN** | **Deep Neural Network** |
| **MLP** | **Multi Layer Perceptron** |
| **LR** | **Logistic Regression** |
| **DT** | **Decision Tree** |
| **NB** | **Naive Bayes** |
| **SVM** | **Support Vector Machine** |

| | |
|---|---|
| **KNN** | **K Nearest Neighbor** |
| **AB** | **Adaptive Boosting** |
| **RF** | **Random Forest** |
| **GB** | **Gradient Boosting** |
| **XGB** | **Extreme Gradient Boosting** |
| **ANN** | **Artificial Neural Network** |
| **RBF** | **Radial Basis Function** |
| **SMOTE** | **Synthetic Minority Oversampling Technique** |
| **WF** | **Workload Forecasting** |
| **SDP** | **Software Defect Prediction** |
| **FL** | **Federated Learning** |
| **HAR** | **Human Activity Recognition** |
| **AI** | **Artificial Intelligence** |
| **HPT** | **Hyperparameter Tuning** |
| **GWO** | **Grey Wolf Optimizer** |
| **WOA** | **Whale Optimization Algorithm** |
| **CFP** | **Cloud Failure Prediction** |
| **AUC** | **Area Under the Curve** |
| **RNN** | **Recurrent Neural Network** |
| **FedAvg** | **Federated Averaging** |
| **RQ** | **Research Question** |
| **RO** | **Research Objective** |
| **WMC** | **Weighted Methods per Class** |
| **DIT** | **Depth of Inheritance Tree** |
| **CBO** | **Coupling Between Objects** |
| **NOC** | **Number of Children** |
| **DIT** | **Depth of Inheritance Tree** |

| | |
|---|---|
| **NPM** | **Number of Public Methods** |
| **RFC** | **Response for a Class** |
| **LCOM** | **Lack of Cohesion of Methods** |
| **LOC** | **Lines of Code** |
| **NR** | **Number of Revision** |
| **NDC** | **Number of Distinct Committers** |
| **NMC** | **Number of Modified Lines** |
| **NDPV** | **Number of Defects in Previous Version** |
| **CFP** | **Cloud Failure Prediction** |
| **MSE** | **Mean Squared Error** |
| **RMSE** | **Root Mean Squared Error** |
| **MAE** | **Mean Absolute Error** |
| **MAPE** | **Mean Absolute Percentage Error** |
| **TP** | **True Positive** |
| **TN** | **True Negative** |
| **FP** | **False Positive** |
| **FN** | **False Negative** |
| **TPR** | **True Positive Rate** |
| **FPR** | **False Positive Rate** |

# Chapter 1

# INTRODUCTION

## 1.1 Introduction

Cloud Computing (CC) has emerged as a transformative paradigm by enabling dynamic and on-demand provisioning of computing resources over the internet. It offers scalability, elasticity, and cost-efficiency to both enterprises and research environments. A key enabler of CC is virtualization, which abstracts physical hardware and allows multiple workloads to run concurrently with improved resource utilization. Without virtualization, the implementation of CC infrastructure would be highly inefficient, as it would require dedicated physical resources and increased hardware maintenance overhead [119]. Traditionally, hypervisor-based virtualization (using virtual machines (VMs)) has been the backbone of cloud platforms. However, a more lightweight and efficient form of virtualization has recently gained prominence—container-based virtualization. Unlike VMs that encapsulate a full guest operating system, containers share the host system's kernel and isolate applications at the process level. This makes them faster to start, more portable across environments, and more efficient in resource usage. As a result, containers are increasingly replacing virtual machines in cloud-native application deployment due to their fast launch time, scalability, platform independence, and smaller resource footprint.

### 1.1.1 What is Container?

A container is a standardized unit of software that packages up code along with all its dependencies—such as libraries, binaries, and configuration files—so that the application runs quickly and reliably across various computing environments [193]. This encapsulation ensures consistency between development, testing, and production environments, thereby improving portability and productivity.

Containerization refers to the process of running multiple isolated applications on the same host operating system. Unlike virtual machines that each require a separate guest OS, containers share the host OS kernel, significantly reducing overhead. As a result, containers are lightweight, faster to deploy, and smaller in size, making them highly suitable for microservices architectures and distributed systems. Figure 1 shows the key architectural differences between traditional hypervisor-based virtualization and container-based virtualization. On the left, each virtual machine runs a separate guest operating system over a hypervisor, leading to higher resource usage and slower performance. In contrast, the right side shows containers sharing the host OS kernel via a container engine (e.g., Docker), significantly reducing overhead. This shared approach improves scalability, start-up speed, and resource efficiency, making containers more suitable for cloud-native and microservices-based environments.

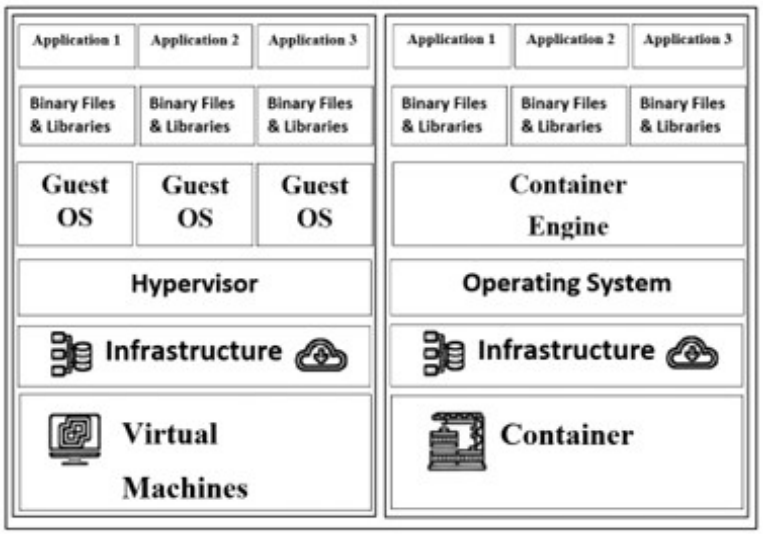



Figure 1.1: Architectural comparison between virtual machine and container

### 1.1.2 What is Docker, Docker image, and Dockerfile?

Docker is the most widely used open-source containerization platform, designed to simplify the creation, deployment, and management of containerized applications. In the Docker ecosystem:

- **Docker Image:** A read-only template containing instructions for creating a Docker container. It includes the application code, runtime environment, libraries, and dependencies.
- **Docker Container:** A runnable instance of a Docker image, encapsulated with its own filesystem and process space.
- **Dockerfile:** A script consisting of a set of instructions used to build Docker images automatically.

Docker streamlines application delivery and supports DevOps pipelines by ensuring uniform behavior across various platforms. It also enables versioning, reusability, and automation. The primary advantage of containers is their speed and efficiency, as they start within seconds and consume fewer resources. However, one limitation is that containers are typically less secure than virtual machines because they share the host OS kernel.

## 1.2 Containerization Technologies

Several container technologies exist, each targeting different domains such as CC, scientific workloads, high-performance computing, and edge environments. Some of the leading containerization platforms include:

- **Docker:** The industry-standard platform for container creation and orchestration. It evolved from Linux Containers (LXC) and provides functionalities such as layered image reuse, automated builds, and container versioning.

- **LXC (Linux Containers):** A low-level container runtime that uses Linux kernel features (namespaces and cgroups) to isolate processes and network spaces. LXC allows multiple isolated Linux systems to run on a single host.
- **OpenVZ (Open Virtuozzo):** A containerization solution for Linux that shares a common kernel among all containers. It is similar to LXC but focuses more on performance and density in Linux environments.
- **Singularity:** A secure container solution tailored for high-performance computing (HPC). It allows users to build containers that are portable across computing clusters. It supports building images from scratch or converting existing Docker images. Singularity emphasizes execution by unprivileged users, enhancing security.
- **uDocker:** A lightweight, rootless container execution environment designed for shared clusters where administrative access is restricted. It runs Docker containers without requiring Docker engine or root privileges, making it suitable for academic and research infrastructure.

These technologies execute their own methods to achieve network and process isolation, ensuring performance and security while running multiple containers on a single host.

## 1.3 Container Orchestration Tools

Container orchestration becomes essential as the scale and complexity of container deployments grow. Orchestration tools automate tasks such as container scheduling, resource allocation, fault tolerance, scaling, service discovery, and communication between containers [27] [6]. They play a critical role in managing distributed and microservices-based applications in production.

Common container orchestration platforms include:

- **Docker Swarm:** A native orchestration tool for Docker that manages clusters

of Docker engines. It supports automated container deployment, service scaling, and load balancing across nodes.

- **Kubernetes:** The most widely adopted orchestration system, originally developed by Google. Kubernetes enables automatic container deployment, horizontal scaling, self-healing, and rolling updates. It supports application reproducibility and platform portability, making it ideal for managing scientific workloads and cloud-native applications.

- **Apache Mesos:** A cluster manager that abstracts CPU, memory, storage, and other resources and allocates them to applications. Mesos supports both LXC and Docker containers and integrates with third-party orchestrators such as Kubernetes for service discovery.

- **Nextflow:** A domain-specific language and workflow management system designed for the reproducible execution of computational pipelines, especially in genomics and life sciences. It can run on local machines, HPC clusters, and cloud platforms, making it highly portable and scalable.

- **OpenShift:** An enterprise-grade container orchestration platform built on Kubernetes, offering enhanced security, developer tools, CI/CD integration, and policy-based governance.

- **Rancher:** A complete container management solution that provides a unified interface for deploying and managing Kubernetes clusters across cloud and on-premise infrastructure.

These tools help maintain application availability, enable rapid scaling, monitor system health, and automate recovery in case of failure. Container technology has been used in many fields such as scientific computing, big data processing, high-performance computing, grid computing, high throughput computing, CC, the internet of things, fog and edge computing, and DevOps.

## 1.4 Phases in Maintenance of Software Containers

The maintenance of software containers involves a series of systematic phases that ensure the reliable, secure, and efficient operation of containerized applications throughout their lifecycle. These phases are crucial for sustaining the performance, security, and scalability of container based systems in both development and production environments. There are various phases in maintenance of software containers which are as follows:

### 1.4.1 Container Image Detection

This phase mainly focuses on smell detection, image mining, and image tagging approaches in the context of containerizing applications. These approaches contribute to identifying and validating container images to ensure quality, security, and maintainability. Smell detection uncovers anti-patterns or bad practices within container configurations through static analysis (e.g., analyzing Dockerfiles) and dynamic analysis (e.g., monitoring runtime behavior) [165] [312] [180]. Issues such as bloated images, hardcoded credentials, improper user configurations, and inefficient resource usage are detected. Image mining [281] extracts metadata, usage patterns, and versioning information from registries, while tagging [165] [318] [44] aids in classifying and managing images systematically. Together, these methods provide a comprehensive strategy for improving the overall quality and trustworthiness of containerized applications.

### 1.4.2 Container Scheduling

Container scheduling refers to the automated process of deploying containers across available nodes in a cluster based on time, resource availability, energy consumption, load utilization, cost, network bandwidth, affinity rules, and scalability needs. Efficient scheduling is crucial for optimal performance and resource utilization. Various types of performance objectives such as load balancing, resource utilization, makespan, en-

ergy consumption, scalability, cost, etc. have been used in the literature to analyze the performance of container scheduling techniques. A survey done by Gill and Buyya [82] examine domain specific optimization objectives. An optimization objective is used to access quality of the solution provided by an algorithm. Optimization objectives can be single objective or multi-objective depending on the optimization required for the algorithm. Kubernetes and Docker Swarm orchestration platforms consist of three default container scheduling approaches to deploy the containers on computing nodes. Random strategy schedules the containers on any node randomly, binpack strategy schedules the containers on the node containing the largest number of containers, and spread strategy schedules the containers on nodes containing the least number of containers. In the literature, authors have used various heuristic [246] [201], meta heuristic [210] [271], mathematical modelling [22] [86], and machine learning (ML) techniques [48] [219] which have been used to schedule the containers on working nodes.

Efficient container scheduling in cloud environments must account not only for resource availability but also for potential task and job failures due to hardware faults, resource contention, or system anomalies. Cloud failure prediction (CFP) involves using ML or statistical models to anticipate task failures by analyzing historical job execution data, system logs, and performance metrics [275]. By predicting failures in advance, the scheduler can proactively reschedule tasks, migrate containers, or adjust resource allocations to maintain system reliability and minimize downtime. In parallel, workload balancing ensures that tasks are evenly distributed across nodes to avoid overloading any single resource. Modern container orchestrators like Kubernetes support workload balancing strategies using affinity/anti-affinity rules, taints and tolerations, and resource-based bin packing. Incorporating failure prediction into scheduling logic enhances fault tolerance, while effective workload balancing improves overall system throughput and energy efficiency.

### 1.4.3 Security Analysis of Containers

As containers follow kernel sharing property, there is a need to evaluate container security measures. Security analysis involves scanning container for identifying security vulnerabilities, misconfiguration, and insecure dependencies in containers [119], approaches to mitigate the vulnerabilities [178] [152], security attacks, and approaches to detect the security attacks [175] [167] [299]. It includes checking for known CVEs (Common Vulnerabilities and Exposures), analyzing user privileges, and monitoring for abnormal behavior at runtime. Tools such as AppArmor [72], Anchore [61], Microscanner [206], Clair [70], Cilium [51], JFrog Xray [260], Dagda [85], Aqua Trivy [283], Qualys [60], Snyk [304] are commonly used to ensure containers adhere to security best practices. Scanning official and verified images from Docker Hub helps identify latent vulnerabilities or outdated packages that could pose security risks. This phase also includes best practices such as implementing least-privilege principles, using non-root users and isolating containers via namespaces and cgroups. Overall, security analysis ensures that containers remain trustworthy and resilient in production environments.

### 1.4.4 Container Performance Evaluation Analysis

Container performance evaluation analysis focuses on assessing the runtime efficiency and resource utilization of containerized applications. As containers are deployed in large-scale and resource-constrained environments, it is essential to monitor key performance metrics such as CPU usage, memory consumption, I/O throughput, network latency, and container start-up time. In the literature, authors have proposed various frameworks and techniques to assess the performance of containers [29] [193] [236] [320] [176]. Additionally, several benchmark tools have been used in the literature to evaluate container performance [293]. Benchmark tools are broadly classified into four categories: CPU intensive benchmark (Y Cruncher [315], Linpack [169], Geekbench [79], Sysbench [266]), memory intensive benchmark (STREAM [264], RAM

speed/SMP [84]), disk i/o intensive benchmark (Bonnie++ [35], Sysbench [266], Iozone [109]), and network i/o intensive benchmark (Netperf [226], Iperf [110]). There exist various container monitoring tools that access metrics such as CPU utilization, memory utilization, disk i/o utilization, network bandwidth, CPU limit, and memory limit [36]. Sematext [58], Docker stats [71], Prometheus [231], Dynatrace [56], cAdvisor [211], Scout [59], and Sensu [251], etc. are some existing container monitoring tools. This analysis helps identify bottlenecks, optimize resource allocation, and ensure service-level objectives (SLOs) are met. Additionally, this phase may involve stress testing,scalability analysis, and runtime benchmarking to ensure containers remain responsive under varying workloads. Through systematic performance analysis, developers and system administrators can enhance system reliability, reduce cost, and improve user experience.

## 1.5 End-to-End Predictive Modeling and Deployment

End-to-End Predictive Modeling and Deployment refers to the holistic process of designing, developing, and operationalizing predictive models—from initial data exploration and model building to deploying trained models within real-world environments where they can generate actionable predictions.

### 1.5.1 Key Phases of the Process

Figure 1.2 represents the end-to-end lifecycle of my proposed framework. The following are the various key phases involved in the end-to-end predictive modelling and deployment process:

1. **Problem Definition**

   Clearly articulate the research objectives by identifying specific prediction tasks (e.g., failure prediction, workload forecasting, software defect prediction) and understanding the contextual constraints influencing the problem.

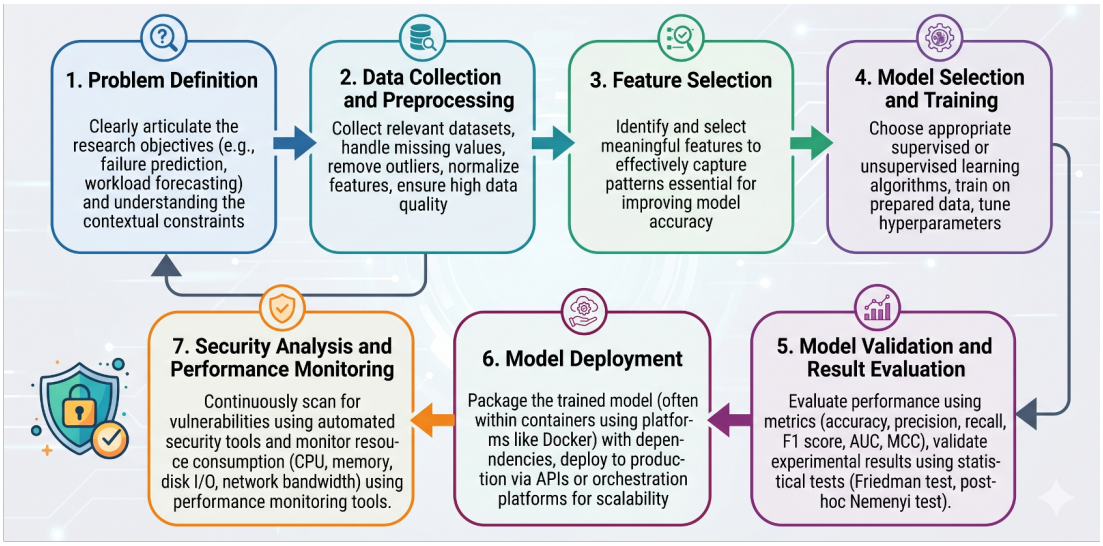



Figure 1.2: End to End Predictive Modeling and Deployment

2. **Data Collection and Preprocessing**

   Collect relevant datasets that capture historical events and preprocess the data by handling missing values, removing outliers, normalizing features, and ensuring high data quality, thereby establishing a reliable foundation for effective modelling.

3. **Feature Selection**

   Identify and select meaningful features from the raw data that effectively capture patterns essential for improving model accuracy.

4. **Model Selection and Training**

   Based on the problem requirements, choose appropriate supervised or unsupervised learning algorithms. Train the model on the prepared data while tuning hyperparameters to optimize learning performance.

5. **Model Validation and Result Evaluation**

   Evaluate the model's performance on test datasets using metrics such as accuracy, precision, recall, F1 score, AUC, and Matthews correlation coefficient (MCC). Further validate the experimental results using statistical tests like the Friedman test and post-hoc Nemenyi test to ensure robustness.

6. **Model Deployment**

   Package the trained model—often within containers using platforms like Docker—with all necessary dependencies. Deploy the containerized model to production environments, enabling prediction services via APIs or integration with orchestration platforms for scalable operation.

7. **Security Analysis and Performance Monitoring**

   Continuously scan deployed model containers for known vulnerabilities, misconfigurations, insecure dependencies, and compliance issues using automated security tools. Simultaneously, monitor resource consumption metrics (CPU, memory, disk I/O, network bandwidth) using performance monitoring tools to ensure efficient and reliable operation.

By incorporating security vulnerability assessment and resource usage analysis as integral steps in the predictive modelling lifecycle, this approach delivers not only accurate and automated predictions but also resilient, secure, and efficiently managed deployments—crucial for production-grade machine learning in modern, containerized cloud environments.

## 1.6 Literature Survey

This section presents a comprehensive review of existing studies that are recent and related to CFP, workload forecasting (WF) in CC environment, software defect prediction (SDP), human activity recognition (HAR), and security and resource utilization analysis of docker containers. Each domain contributes uniquely in developing intelligent, scalable, and secure software systems.

- **Cloud Failure Prediction and Workload Forecasting**

As cloud infrastructures become increasingly dynamic and complex, challenges related to resource utilization, WF, and failure management have gained prominence.

Accurately predicting workload and failures in CC environment helps in maintaining resource utilization, improving availability, and reducing operational costs. Recent advancement in ML and deep learning (DL) have significantly improved predictive accuracy in WF and CFP. These models have been evaluated on widely used datasets such as Alibaba cluster trace, Google cloud trace (GCT), trinity, mustang, and bitbrains dataset, representing diverse workload characteristics.

- **Workload Forecasting**

Bi *et al*. [31] introduced BGLSTM, a combined model of BiLSTM and GridLSTM, demonstrating an improvement in accuracy as compared to traditional DL models. Karim *et al*. [126] proposed BHyPreC, hybrid model of BiLSTM, LSTM, GRU, and 1d cnn layer for workload prediction. Google cloud trace is used to evaluate the prediction accuracy of proposed model and shown better results as compared to individual techniques. Kumar *et al*. [144] introduced workload prediction model using LSTM networks. They evaluated the prediction performance of the model using three datasets namely HTTP traces of Calgary server, NASA server, and Saskatchewan server. The result shows that the proposed model reduces MSE up to 0.00317. Similary, Wang *et al*. [295] presented the CNN-BiGRU attention model, showing a 37.4% improvement in prediction results compared with baseline methods. Xu *et al*. [314] introduced es-DNN, an efficient supervised DNN algorithm, achieving a 15% reduction in MSE. Zhang *et al*. [331] proposed a stacked autoencoder model based on canonical polyadic decomposition to predict the virtual machines workload. They evaluated the prediction performance on the PlanetLab dataset, demonstrating that the proposed model reduces MAPE and RMSE while achieving high training speedup. Zhang *et al*. [327] introduced a combined model of LSTM and triple exponential smoothing, effectively reducing MAPE compared with existing approaches. Patel and Kushwaha [233] presented a hybrid model of CNN and LSTM which is termed as pCNN-LSTM, demonstrating that pCNN-LSTM outperforms LSTM, BiLSTM, CNNLSTM, and CNNBiLSTM.

- **Cloud Failure Prediction**

Asmawi *et al*. [275] presented a comparative analysis of various ML (LR, DT, RF, GB, and XGB) and DL (single layer LSTM, and Bi layer LSTM) algorithms, which shows that XGB performs better in case of job failure whereas DT and RF perform better in case of task failures. Gao *et al*. [77] introduced a model based on BiLSTM, achieving an accuracy of 87% for job failure and 93% for task failure prediction. Jassas and Mahmoud [113][114] developed ANN-based failure prediction models, outperforming RF and DT for large datasets. They further enhanced model interpretability using feature selection techniques such as SelectKBest, RFE, and feature importance with RF. Vani and Sujatha [287] presented a model which is a combination of catboost algorithm (for feature selection) and hyperparameter tuned MLP (for classification). Bommala *et al*. [34] implemented various ML algorithms (RF, DT, KNN, NB, QDA, GB, and XGB) combining with recursive feature elimination (RFE) for failure prediction. They used GCT 2011, mustang, and trinity dataset for evaluating the performance of the algorithms. The result shows that RF and DT combining with RFE gives better performance as compared to other algorithms. Islam and Manivannan [111] presented LSTM based failure prediction model, achieving an accuracy of 91% for job failures and 87% for task failures. Chowdary *et al*. [50] presented a comparative analysis of hidden Markov model, hybrid model of CNN and LSTM, and BiLSTM, showing that BiLSTM outperforms other approaches with an accuracy of 93.9%, precision of 0.941, and f1 score of 0.922. Tuns and Spataru [285] implemented 4 ML approaches (GB, DT, LR, and RF) on google' borg cluster workload trace. They analyzed that GB outperforms other approaches for job failures with an accuracy of 83.97% and DT outperforms other approaches for task failures with an accuracy of 98.79%. Sayed *et al*. [250] developed a ML based framework (RF) for job and task failure prediction. They also presented how the prediction model can be used to reduce the job and task failures. They used GCT 2011, CMU OpenCloud Hadoop cluster, and LANL HPC Cluster trace to perform the experiment. Shetty *et al*. [254] developed a model utilizing XGB classifier for predicting task failures. They used random oversampling, random oversampling, and SMOTE to balance the dataset. GCT 2011 dataset is used in this study to perform the experiment. The result shows that XGB classifier with

SMOTE technique performs better with recall of 94.8% and a precision of 92%.

The reviewed literature reveals that while ML, DL and hybrid neural architectures have greatly advanced workload forecasting and failure prediction accuracy, challenges remain concerning privacy preservation, model scalability, and secure deployment.

- · **Software Defect Prediction**

SDP has evolved from traditional ML/DL models to more advanced techniques such as FL, optimization, and secure deployment. Early approaches focused on models trained on datasets like NASA, AEEEM, ReLink, and SOFTLAB, showing that algorithms such as SVM, Naïve Bayes, and CNNs could capture defect-prone modules but faced limitations in scalability and privacy. Komalasari and Candra [140] developed a classification model using a combination of product, process, and project metrics. They applied NB, LR, and RF techniques to predict bugs, finding that integrating these three metrics yielded effective results, particularly with the RF model. Akbar *et al.* [12] combined GWO and PSO for SBP and evaluated their hybrid approach on the NASA MDP dataset using XGBoost, Catboost, and LightGBM classifiers. AUC-based analysis confirmed that the hybrid optimization strategy outperformed individual metaheuristic techniques. Khleel and Nehej [134] presented a model which is a combination of BiLSTM and oversampling approaches (Random oversampling and SMOTE). They utilized open-source dataset from publicly available PROMISE repository. The result depicts that proposed model performs better as compared to state of the art methodologies in terms of accuracy, precision, recall, AUC, and MCC. Malhotra and khan [191] presented a model based on 2 phase GWO combining with SMOTE. They applied the proposed model to the NASA, AEEEM, ReLink, and PROMISE repository datasets. They found that RF paired with proposed 2 phase GWO model outperforms existing approaches in terms of AUC metrics. Gupta *et al.* [90] discussed about the importance of HPT in building effective DNN model. They used NASA dataset (PC1, PC2, KC1, and KC3). The proposed model's prediction performance is examined in respect of precision, accuracy, recall, as well as f1 score. They analyzed the result based on dropout rate. Their result shows that the DNN with dropout rate achieves better per-

formance as compared to DNN without dropout rate. Choudhary *et al.* [49] introduced new change metrics and evaluated their impact alongside existing metrics on SBP using the eclipse dataset. They utilized DT, KNN, and RF classifiers, assessing performance with precision, recall, and F-measure metrics. Their analysis revealed that a combined model of change and code metrics outperformed models relying solely on individual metric sets for the eclipse dataset. Zhu *et al.* [341] introduced a model, IMDAC, based on DL and soft computing approaches. For data augmentation, InfoGAN is used, and for feature selection, MSEA (multi objective optimization algorithm) is used. For prediction, hybrid model of DAE and CNN is used. They applied proposed model to the AEEEM, NASA, ReLink, PROMISE, and SOFTLAB repository datasets. They found that proposed model outperforms existing approaches in terms of f1 score, AUC, MCC, and G-measure. Prashanthi and Chandra Mohan [239] presented optimized neural network (NN) model for predicting the bugs. They used relief algorithm for selecting the important features and the hyperparameters of NN are optimized using hybrid of GWO and social spider algorithm (SSA). The result of the proposed model indicates that it performs better than the existing methodologies with an accuracy of 93.64%, precision of 99%, f1 score of 93.53%, sensitivity of 95.14%, and specificity of 99%. Chen *et al.* [43] proposed dual ensemble model, DESDP, combined with neural network. Dataset from NASA MDP repository is utilized to evaluate the predictive efficacy of the proposed model. They concluded that the proposed model outperforms the base models in terms of AUC, f1 score, and MCC. These studies mainly focuses on ML, DL, HPT of ML and DL models for SDP. Recent studies highlight the effectiveness of FL for SDP.

Chen *et al.* [42] employed MobileViT in an FL setup, reducing communication costs while improving G-mean and AUC. Wang *et al.* [292][291] proposed FTLKD and FRLGC, leveraging knowledge distillation and reinforcement learning, which consistently outperformed traditional methods across datasets. Furthermore, Wang *et al.* [290] also developed a federated sparse compression algorithm to minimize communication overhead during the model training. They evaluated the predictive efficacy of this algorithm on the AEEEM, NASA, and Relink datasets and integrated differential privacy and capsule networks to improve security and representation learning. Their

results demonstrated that there is an improvement in G-mean and AUC score while communication cost reduces significantly. Yamamoto *et al*. [316] introduced FLR with LR, achieving higher AUC and precision, while Zhou *et al*. [340] combined FL with knowledge distillation for improved accuracy. Deployment-focused works addressed security and resource efficiency. Malhotra *et al*. [189] proposed FLCNN, a Dockerized FL-CNN model, which achieved better AUC and MCC while ensuring secure and efficient deployment. Similarly, Nandi *et al*. [220] introduced Docker-enabled FL for distributed tasks, achieving strong accuracy and F1 scores with low overhead.

Overall, literature shows that combining FL with optimization and secure containerized deployment strategies leads to models that are more accurate, privacy-preserving, and scalable, making them suitable for real-world SDP in large-scale environments.

- · **Human Activity Recognition**

HAR leverages sensor or video data to classify actions such as walking, running, or sitting. DL models have revolutionized HAR by automating feature extraction and temporal sequence modeling.

Luwe *et al*. [181] introduced a combined model of 1D CNN and Bidirectional LSTM for wearable sensor based HAR, achieving an accuracy of 95.48% on UCI-HAR, 94.17% on Motion Sense, and 100% on single accelerometer dataset. Bhattacharya *et al*. [30] presented an ensemble of CNN, CNNLSTM, ConvLSTM, and Stacked LSTM, attaining competitive results across WISDM, UCI-HAR, and PAMAP2 datasets. Challa *et al*. [39] introduced CNN-BiLSTM model, achieving an accuracy of 96.05% on WISDM, 96.37% on UCI-HAR, and 94.29% on PAMAP2. Xia *et al*. [309] presented a combined model of CNN and LSTM, demonstating an accuracy of 95.85% on WISDM, 95.78% on UCI-HAR, and 92.63% on Opportunity dataset. Kobayashi *et al*. [139] introduced MarNASNets (mobile aware CNN by using neural architecture search), achieving an accuracy of 92.60% on HASC, 94.52% on UCI-HAR, and 88.92% on WISDM datasets. Wang *et al*. [298] presented DMEFAM, a deep multi-feature extraction framework integrating BiGRU for temporal attention and ResNet-18 for spatial attention, achieving an accuracy of 99.2% on DAAD, 97.9 on WISDM, and

96% on UCI HAR dataset. Zhao *et al*. [336] proposed Residual BiLSTM and evaluated the effectiveness of the proposed model using UCI HAR dataset, demonstrating that the proposed model attains 93.6% accuracy rate and 93.5% F1 score

Taken together, these studies demonstrate the effectiveness of CNN, LSTM, and hybrid models for HAR. However, most works employ single architectures without systematically exploring stacking-based ensembles and federated learning (FL). Furthermore, research on the FLAAP dataset is still in its infancy and has not leveraged DL methods. This leaves ample scope for privacy-aware, deployment-ready HAR models leveraging containerization and edge AI.

- · **Security Vulnerability and Resource Consumption Analysis**

With the widespread adoption of containerization technologies such as Docker, ensuring the security and efficiency of deployed models has become a critical concern. As Docker continues to gain prominence across various application domains, the need for effective monitoring and mitigation of vulnerabilities in Docker images has increased significantly. Recent studies have investigated security vulnerabilities in Docker images and containerized systems, along with analyzing their resource consumption under different workload conditions.

Efe *et al*. [10] performed a detailed review of existing studies to identify the most common CVEs, methods to detect vulnerabilities, methods to detect DoS attack, and how to prevent docker images from DoS attack. Gummaraju *et al*. [88] analyzed security vulnerabilities in docker hub official images. They identified that there exist more than 30% of the official images which contains high severity vulnerabilities. Zerouali *et al*. [324] analyzed the relationship between vulnerabilities in containers, outdatedness of containers and buggy packages installed in containers. They used technical lag to check the outdatedness of containers. They identified that all studied Debian container have vulnerabilities and buggy packages. Shu *et al*. [256] introduced DIVA framework to analyze security vulnerabilities in official and community docker images. They identified that many images are outdated and there exist more than 180 security vulnerabilities on average in docker images. Zerouali *et al*. [323] analyzed

that the vulnerabilities in javascript packages might have negative impact on docker hub images. Zerouali *et al.* [322] presented ConPan, an automated tool to monitor and analyze outdatedness, vulnerabilities, and bugs. This tool can be used as a CLI or integrated with other processes through its python API. Gholami *et al.* [81] analyzed the package updation in docker images. They shows that there is a median of 8.6 upgradation of packages and 2.1 downgrades of packages per docker image. Wenhao and Zheng [303] analyzed docker architecture and potential security risks in docker containers. Huang *et al.* [105] performed a detailed analysis of existing security mechanism and threats for containers. They also introduced a framework for detecting the threats in docker containers. He *et al.* [98] conducted a thorough and extensive survey to explore diverse attacks on DL models. Within their research, they divided the types of attacks into four categories: model extraction attack, poisoning attack, model inversion attack, and adversial attack. Xiao *et al.* [310] conducted an empirical investigation aimed at identifying vulnerabilities and assessing their associated risks within three DL frameworks (Caffe, Torch, and TensorFlow). The risks that have been highlighted include the possibility of denial of service, system compromise, and evasion of detection. Harzevili *et al.* [95] analyzed the vulnerabilities in mlpack, pandas, scipy, numpy, tensorflow, pytorch, and scikit-learn. They collected 683 vulnerabilities and found the type of vulnerability, cause of vulnerability, how to fix the vulnerability, and symptoms of the vulnerability. Javed and Toor [115] examined the effectiveness of different container scanning tools by assessing their coverage and accuracy.

The resource usage of containers was investigated by Morabito *et al.* [213] using different benchmark tools such as linpack and ycruncher for CPU utilization, stream for memory allocation, bonniee++ for disk utilization, and netperf for network traffic consumption. Kozhirbayeu and Sinnott [141] examined the resource usage using open-source benchmark tools such as geekbench, linpack, and ycruncher for CPU usage, stream for memory allocation, disk utilization using bonniee++ and random write speed, and network traffic consumption using netperf and iperf. A monitoring setup was introduced by Matti Holopainen [101] for tracking the resource utilization (CPU utilization, memory allocated, and network bandwidth consumption) within the con-

tainer environment. This setup utilizes Prometheus and Grafana, with cAdvisor and node exporter serving as data acquisition tools. Tiia Leppänen [156] conducted a study to examine and make a comparison between the monitoring tools Prometheus and Grafana with Zabbix and Libre NMS. The study utilizes Node exporter and SNMP exporter for the purpose of collecting the data. Malhotra *et al.* [189] conducted an extensive analysis of security issues and resource consumption in docker based FL framework for SDP.

Collectively, these works underscore the importance of combining security auditing, vulnerability analysis, and performance monitoring in modern containerized ML deployments.

The reviewed literature across all five domains highlights rapid advancements in ML, DL, and FL-based predictive intelligence. While substantial progress has been made in accuracy and efficiency, significant research gaps persist in the areas of privacy preservation, cross-domain generalization, container-level security, and scalable distributed learning. Addressing these challenges forms the central motivation for this thesis, which proposes federated, optimized, and containerized frameworks for WF, CFP, SDP, and HAR, coupled with detailed analysis of security vulnerabilities and resource utilization in Dockerized environments.

## 1.7 Motivation and Objective of the Thesis

As ML models become increasingly integral to modern applications, ensuring their reliable and secure deployment is critical. This research focuses on an end-to-end approach that encompasses the entire lifecycle of predictive modeling—from data preparation and model development to deployment within containerized environments. Containerization technologies, such as Docker, enable scalable, consistent, and portable deployment of ML models across various computing platforms, addressing compatibility and operational challenges.

However, deploying predictive models in containers introduces unique concerns re-

lated to security and performance. Containerized models must be rigorously assessed for vulnerabilities—including outdated dependencies, misconfigurations, and potential access control weaknesses—to safeguard the integrity and confidentiality of prediction services. Automated security scanning tools and continuous monitoring are vital components of this protective framework.

Equally important is the performance evaluation of deployed containerized models. Real-time monitoring of resource utilization and response metrics ensures that models operate efficiently, meet service-level objectives, and can scale dynamically in production environments. Integrating performance analysis tools facilitates proactive identification of bottlenecks and informs optimization strategies.

By addressing predictive modeling, secure container deployment, and performance monitoring in a unified framework, this thesis aims to contribute robust methodologies to maintain high-quality, trustworthy, and efficient ML services. This holistic perspective reflects the growing need for practical and resilient artificial intelligent (AI) systems in production, forming the foundation for the objectives and contributions outlined in this thesis.

### 1.7.1 Vision

The thesis envisions building secure, scalable, and trustworthy AI-driven models for software engineering by leveraging FL, DL, and container-based virtualization. It addresses critical challenges such as data privacy, distributed training, cloud workload management, and secure deployment, while ensuring operational feasibility through resource utilization and security vulnerability analysis.

### 1.7.2 Focus

The focus of this thesis is centered on addressing key challenges in the maintenance of software containers through a series of well-defined research objectives (RO). These objectives aim to develop and evaluate predictive models, deploy them using Docker

containers, and investigate both security vulnerabilities and resource utilization in containerized environments. The research further focuses on building efficient classification models for tasks such as SDP, HAR, WF, and CFP (job and task failures).

Through a combination of FL, DL, and container-based virtualization, the thesis seeks to improve the reliability, scalability, and trustworthiness of AI-driven software systems. The models and frameworks developed in this research aim to support cloud-native development practices and aid software practitioners in making data-driven, resource-efficient decisions. This study explicitly addresses the following research objectives:

RO1: To perform a systematic literature review to understand existing methodologies and frameworks used across various phases of software container maintenance.

RO2: To explore and analyze security vulnerabilities and threats associated with software containers.

RO3: To develop a novel predictive model for cloud workload forecasting using a federated learning approach.

RO4: To analyze the effect of hyperparameter tuning algorithms and develop predictive models for predicting task failures and job failures using federated learning and deep learning techniques.

RO5: To explore and exploit performance evaluation of software containers using benchmark tools and real time monitoring tools with respect to CPU, memory, network, and disk utilization.

### 1.7.3 Goals

Each of the research objectives outlined above is accompanied by specific goals that detail the steps and milestones necessary to achieve the desired outcomes. These goals are designed to guide the research process, ensuring that each objective is addressed systematically and comprehensively.

#### 1.7.3.1 Systematic Literature Review

- Study of existing research publications that would help to understand the process and procedure of maintenance of software containers. Systematic literature review will help to understand the importance of various phases in software container maintenance.
- Extensive study of existing literature to understand the significance of proposed approaches which are related to container image detection, tag recommendation, and detecting smells in images.
- Study of various ML, mathematical modelling, heuristic, and meta heuristic approaches used in literature for scheduling containers on working nodes.
- Extensive study of existing literature to understand the significance of proposed approaches which are related to security vulnerability detection, threat detection and approaches which are used to resolve security issues.
- Extensive study of existing literature to understand the significance of benchmark tools and monitoring tools in evaluating performance of containers.
- A review of the literature would also help to identify the problem areas and gaps in the existing literature.

#### 1.7.3.2 To explore and exploit security issues (vulnerabilities and threats) in software containers

- Examine the types and sources of vulnerabilities and threats within containerized environments, including dependency issues, outdated libraries, and insecure configurations.
- Utilize a set of open-source vulnerability scanning tools (Anchore, Aqua Trivy, Docker Scout, and Snyk) to analyze Docker images encapsulating DL and FL models.

- Compare the detection efficiency of different tools in terms of accuracy, coverage, and severity classification to establish a reliable vulnerability assessment framework.

- Develop a secure container deployment pipeline integrating continuous scanning and monitoring to support DevSecOps practices.

#### 1.7.3.3 Federated learning based cloud workload forecasting

- Design a FL framework for cloud workload prediction, ensuring data privacy across distributed nodes.

- Implement CNN-based FL models trained on real-world cloud datasets such as Alibaba Cluster Trace v2018, Alibaba cluster trace 2020, GCT 2011, and GCT 2019.

- Perform data preprocessing and feature engineering (e.g., scaling, normalization, time-series generation) to improve model convergence and forecasting accuracy.

- Evaluate and compare the predictive performance of the proposed FL model with traditional CNN and LSTM approaches using metrics such as MAE, MSE, RMSE, MAPE, and $R^2$.

- Demonstrate how FL enhances privacy, scalability, and prediction stability in workload forecasting.

#### 1.7.3.4 Cloud failure prediction using deep learning and hyperparameter tuning

- Develop stacked DL and FL models for predicting job and task failures in cloud data centers using real-world traces (e.g., Google Cluster Trace).

- Implement hyperparameter tuning (HPT) using OPTUNA framework.

- Conduct comparative performance analysis of tuned models (e.g., CNN with and without HPT, ANN with and without HPT) to assess improvement in predictive accuracy.

- Evaluate models using accuracy, precision, recall, F1-score, MCC, and AUC to ensure robustness in real-world scenarios.

#### 1.7.3.5 To analyze software defect prediction using federated learning and secure deployment

- Develop FL-based CNN and ANN models optimized through GWO-based HPT for SDP.

- Deploy the optimized models as Docker containers and evaluate them using multiple security scanners (Anchore, Aqua Trivy, Clair, Snyk, etc.).

- Perform resource utilization analysis (CPU, memory, network traffic, and disk I/O) using cAdvisor, Prometheus, and Grafana.

- Validate that federated and containerized approaches improve both predictive accuracy and security resilience in distributed software environments.

#### 1.7.3.6 To develop and analyze a Dockerized ensemble model for HAR

- Construct a stacking ensemble DL framework (DeepHAR) integrating CNN, LSTM, BiLSTM, CNN-LSTM, and CNN-BiLSTM models with a Random Forest (RF) meta-learner.

- Evaluate the proposed framework on WISDM, UCI-HAR, and FLAAP datasets for activity classification using AUC and MCC metrics.

- Perform statistical validation using Friedman and Nemenyi tests to confirm the superiority of the proposed ensemble model.

- Deploy the final model in a Docker environment and conduct security vulnerability and performance monitoring to assess reliability, scalability, and efficiency.

#### 1.7.3.7 To evaluate performance and resource utilization of Docker containers

- Assess the runtime efficiency of containerized DL and FL models using benchmark and real-time monitoring tools.
- Measure key performance indicators, including CPU utilization, memory allocation, disk throughput, and network bandwidth usage.
- Compare the monitoring results obtained from tools like docker stats, cAdvisor, Prometheus, and Grafana.
- Formulate an integrated performance and security evaluation framework for containerized AI systems to ensure scalability, resilience, and operational trustworthiness.

In summary, the goals of this thesis collectively contribute toward developing a holistic, end-to-end framework for intelligent, privacy-preserving, and secure ML model development and deployment. Through FL, optimization, and containerization, this research seeks to advance the fields of CC, software engineering, and AI-driven automation, bridging the gap between model innovation and real-world deployment.

## 1.8 Overview of the Work

The rapid advancement of software systems and CC has introduced significant challenges in software reliability, WF, failure management, and secure deployment. Traditional approaches often struggle to meet the demands of scalability, privacy preservation, and trustworthy deployment. Containers—leveraging technologies such as Docker, LXC, OpenVZ, Singularity, and uDocker—have emerged as central enablers of modern DevOps pipelines, providing lightweight deployment and portability. This

section provides an overview of the research conducted in this thesis. A comprehensive systematic literature review was carried out following the guidelines of Kitchenham [138] to gain a thorough understanding of prior research on the various phases of software container maintenance. The review, which examined 145 primary studies published between 2017 and 2022, highlights that container maintenance research remains fragmented: 42% of the studies focus on scheduling, 24% on performance evaluation, 22% on security, and only 12% on image detection.

The review identifies several limitations. While container orchestration tools like Kubernetes, Docker Swarm, and Mesos enable scalable scheduling, adaptive and intelligent allocation methods remain underdeveloped. Security research highlights persistent vulnerabilities, including malware, denial-of-service attacks, unauthorized access, and kernel exploits, yet defense strategies are primarily limited to static and dynamic analysis alongside vulnerability scanning tools such as Anchore, Clair, and Aqua Trivy. Performance evaluations predominantly emphasize metrics like resource utilization, often neglecting critical factors such as scalability, resilience, and network behavior. Moreover, methods for image detection and tagging are insufficient for assuring secure container supply chains. Collectively, these findings expose gaps in proactive vulnerability auditing, advanced ML powered prediction models, and comprehensive deployment performance monitoring. Situated within this context, the thesis addresses these gaps by proposing a suite of DL based frameworks enhanced with FL, metaheuristic optimization, and deployment-level performance and vulnerability evaluation. Three core research pillars underpin all contributions:

- · Enhancement of predictive accuracy and robustness through DL applied to SDP, WF, and CFP.
- Utilization of optimization and FL paradigms to improve model performance, scalability, and privacy.
- Ensuring secure, resource-efficient deployment via systematic vulnerability scanning and performance monitoring within Dockerized environments.

The first investigation targets security risks in Dockerized environments. Using tools such as Anchore, Aqua Trivy, Docker Scan, and JFrog Xray, a large-scale scan of Docker Hub official and verified images uncovered numerous medium- and high-severity vulnerabilities—including denial-of-service and buffer overflow issues. A key insight was that image popularity metrics such as stars and downloads do not correlate with security levels, emphasizing the imperative for continuous vulnerability auditing. This work established a foundation that balances predictive intelligence with deployment trustworthiness.

Building on this, the thesis develops FL frameworks for SDP. Two models, Federated ANN (FLANN) and Federated CNN (FLCNN), were optimized via the Grey Wolf Optimizer (GWO) for HPT. Experiments on eight large-scale open-source Java projects demonstrated that FLCNN combined with GWO outperformed both centralized and untuned baselines, achieving superior accuracy, AUC, and MCC. To assess practical deployment feasibility, these models were containerized and evaluated for CPU and memory consumption, alongside vulnerability scans. Results revealed the models to be computationally efficient yet still vulnerable to medium-level security risks, highlighting the ongoing challenge of balancing performance and deployment security.

Further advancement came from applying hybrid metaheuristic optimization. Combining the Whale Optimization Algorithm (WOA) with GWO, this approach was applied to CNN-based bug prediction across six open-source Java projects. It consistently achieved the highest accuracy, precision, recall, F1, and MCC scores, outperforming both untuned CNNs and single-optimizer frameworks. These results demonstrate the power of swarm intelligence–based hybrid optimization to elevate predictive performance in software engineering.

Extending the predictive scope to CC environments, the thesis addresses both WF and CFP. A federated CNN framework evaluated on Google and Alibaba cluster traces showed consistent superiority over centralized CNN and LSTM models in accuracy, scalability, and privacy preservation, critical for distributed cloud providers. Concurrently, a CFP framework employs optimized DL models to detect failures proactively.

Validated on real-world datasets, these models achieve strong accuracy, precision, recall, and AUC, enabling early failure identification and enhancing cloud service resilience.

Finally, the thesis demonstrates the cross-domain adaptability of its approaches through HAR. CNN–RNN architectures deployed in Docker achieved high classification accuracy on benchmark datasets. However, vulnerability scans once again revealed persistent medium-level security risks, reinforcing the conclusion that continuous deployment security monitoring is essential regardless of application domain.

In summary, this thesis delivers a unified body of work advancing secure, scalable, and intelligent predictive systems. By integrating DL with FL, swarm-based optimization, and deployment-level performance and vulnerability analysis, it provides both theoretical insights and practical frameworks tailored to software engineering and cloud environments. The fundamental contribution lies not only in augmenting predictive intelligence but also in systematically evaluating how deployable, secure, and resource-aware these systems truly are in real-world contexts. This dual focus ensures the research robustly addresses the challenges of constructing the next generation of resilient, privacy-preserving, and trustworthy software systems.

## 1.9 Organization of the Thesis

This section presents the organization of the thesis. The thesis is structured into eight chapters, each focusing on a specific aspect of the research undertaken to develop and validate techniques to analyze the security vulnerabilities and resource utilization in DL models, to develop and validate DL models for CFP and WF. The organization of the thesis is as follows: **Chapter 1** sets the foundation for the entire research by introducing the fundamental concepts of software containers and outlining the motivation behind the thesis. **Chapter 2** explains the research methodology adopted to achieve the research objectives. **Chapter 3** presents a comprehensive systematic literature review conducted according to established guidelines to identify the research gaps. **Chapter 4** presents the novel FL model for workload prediction in CC environment. **Chapter 5**

proposes a novel DL model integrated with HPT for prediction of job and task failures in CC environment. **Chapter 6** discusses the analysis of security vulnerabilities and resource utilization in predictive models generated for SDP based on FL with HPT. **Chapter 7** presents an analysis of security vulnerabilities and resource utilization in predictive models generated for HAR focusing on ensemble of DL models. **Chapter 8** presents the conclusion, future scope, and social impact of the thesis. A brief description of each chapter is given below:

**Chapter 1:** The introduction sets the stage for the entire research work. It begins by discussing the significance of software containers. It outlines the details of various container orchestration tools, containerization technologies, and the various phases in maintenance of software containers. The chapter also presents the research objectives, the scope of the study, and the contributions of the research. Additionally, it describes the detailed steps involved in end to end life cycle of predictive modeling and deployment.

**Chapter 2:** This chapter describes the research methodology adopted to achieve the ROs. It begins with a discussion on the study's design, including the selection of datasets, the choice of algorithms, the evaluation metrics used to assess model performance, and various tools used to analyze security vulnerabilities and resource utilization. The chapter also covers the experimental setup, data preprocessing techniques, and the steps involved in developing and validating predictive models. Finally, it discusses the systematic approach taken to address the challenges identified in the research.

**Chapter 3:** The third chapter presents a comprehensive systematic literature review conducted according to established guidelines. This review provides a detailed understanding of the existing research in various phases in maintenance of software containers, with a focus on various approaches for container image detection and smell detection, container scheduling techniques including heuristic techniques, meta heuristic techniques, ML techniques, and mathematical modelling techniques, and various techniques and tools used to analyze the security vulnerabilities and resource utilization of

software containers. The review highlights the strengths and limitations of current approaches, identifies research gaps, and sets the foundation for the subsequent chapters.

**Chapter 4:** This chapter presents a FL based approach for cloud workload prediction using real-world datasets Alibaba Cluster Trace 2018, Alibaba cluster trace 2020, GCT 2011, GCT 2019. The proposed model employs a DL based federated framework, where a 1D Convolutional Neural Network (CNN) is used as the base learner. The model's parameters are aggregated using the Federated Averaging (FedAvg) algorithm, and its performance is evaluated using metrics such as MAE, MSE, RMSE, MAPE, and $R^2$. This chapter presents experimental results demonstrating the superiority of FL based CNN model over traditional CNN and LSTM models ensuring privacy and security to the data.

**Chapter 5:** This chapter explores the impact of HPT on the performance of various DL and hybrid models for CFP in a CC environment. A novel stacked ensemble model is proposed, integrating OPTUNA for efficient hyperparameter optimization of both individual DL models and the ensemble itself. The performance of the tuned models is systematically compared with their untuned counterparts and standalone DL models. In addition, the chapter investigates a FL approach, employing a 1D CNN as the base learner to predict cloud failures using the GCT 2011 dataset. Model parameters are aggregated using the FedAvg algorithm, preserving data privacy while enhancing predictive performance.

**Chapter 6:** This chapter explores the use of FL for SDP in a privacy-preserving manner. The proposed model employs CNN and ANN as base learners within the federated framework to ensure data security without centralizing sensitive information. The effectiveness of this approach is compared with traditional ML and DL models. Additionally, the chapter investigates potential security issues and evaluates resource consumption by deploying the FL model in a Docker container environment, ensuring reproducibility, scalability, and efficient deployment. Furthermore, it also discusses the security vulnerabilities identified in official and verified Docker Hub images using various open-source vulnerability scanning tools. Furthermore, this chapter analyzes

the impact of HPT on the performance of a FL based CNN model for SDP. The proposed model incorporates GWO to fine-tune key hyperparameters, including dropout rate, number of epochs, learning rate, and batch size. The performance of the GWO-tuned federated CNN (FLCNN-GWO) model is evaluated and compared against both the untuned FLCNN model and conventional ML and DL approaches. Furthermore, the chapter examines potential security concerns by deploying the proposed model in a Docker container environment, thereby assessing its reproducibility, scalability, and operational efficiency in real-world deployment scenarios.

**Chapter 7:** This chapter presents a novel stacked ensemble model, DeepHAR, for HAR. The proposed model integrates multiple DL architectures—CNN, LSTM, BiLSTM, CNN-LSTM, and CNN-BiLSTM into a unified ensemble framework to improve recognition accuracy. The performance of DeepHAR is evaluated and compared against individual DL models using benchmark datasets including WISDM, UCI HAR, and FLAAP. Furthermore, the proposed model is deployed in a Docker container environment, and its security vulnerabilities are analyzed using a range of open-source vulnerability scanning tools. In addition to accuracy, this chapter also focuses on resource consumption metrics such as CPU utilization, memory usage, network traffic, and disk I/O, offering a comprehensive evaluation of DeepHAR's performance in real-world deployment settings.

**Chapter 8:** The final chapter summarizes the key findings and contributions of the thesis. It discusses the implications of the research for both academia and industry, highlighting the advancements made in software quality prediction. The chapter also identifies potential areas for future research, offering suggestions for further exploration and development in this field.

# Chapter 2

# RESEARCH METHODOLOGY

## 2.1 Introduction

To achieve the objectives of this research and ensure dependable empirical results, a well-organized and systematic approach is essential. The research methodology acts as a blueprint, providing a clear sequence of steps to guide the design, execution, and analysis stages of the empirical experiments. Following a structured methodology enhances consistency, minimizes bias, and strengthens the reproducibility and validity of the findings. This chapter details the research methodology applied in this thesis, outlining a coherent framework aimed at addressing the research questions and validating the proposed approaches. It breaks down each stage of the process, starting from defining the research problem and reviewing the existing literature to identifying relevant variables, analyzing data, and constructing the experimental design. Each step logically builds upon the previous one, ensuring the entire research process is integrated, coherent, and aligned with the overall research goals. The process begins with defining the research problem and reviewing the relevant literature, followed by identifying variables, collecting and preprocessing data, and then using data balancing strategies to handle the class imbalance problem. Special attention is given to the careful development and validation of prediction models. The methodology is further enriched by the inclusion of advanced deployment practices, detailed security analysis,

and rigorous performance assessment using various open-source tools.

The chapter is organized as follows: Section 2.2 describes the empirical research process; Section 2.3 defines the research problem; Section 2.4 presents a comprehensive review of relevant literature; Section 2.5 details the independent and dependent variables considered; Section 2.6 outlines the data analysis methods employed; Section 2.7 discusses the data collection procedures; Section 2.8 elaborates on data preprocessing techniques; Section 2.9 focuses on data balancing approaches; Section 2.10 covers the development and validation of prediction models; Section 2.11 explains performance evaluation metrics; Section 2.12 presents statistical analysis methods used to validate the experimental results; Section 2.13 describes the deployment of prediction models using containerization process; Section 2.14 outlines the tools used to analyze the security vulnerabilities in the model; and Section 2.15 elaborates on the use of performance monitoring and analysis tools for the deployed solutions.

## 2.2 Research Process

The research process consists of a carefully planned sequence of steps designed to comprehensively investigate and address the research problem. This organized approach ensures a smooth and logical progression, guiding every phase of the study from the initial identification of the problem to the final experimental evaluation and conclusions. Figure 2.1 provides a visual overview of the research process, highlighting the stages integrated throughout the thesis chapters to maintain clarity and consistency. Each phase plays a vital role in building a comprehensive and systematic investigation. The subsequent sections offer detailed descriptions of the activities performed at each stage, explain their interconnections, and demonstrate how they collectively contribute to achieving the overall research goals. By following this structured methodology, the research gains coherence, focus, and rigor, thereby strengthening the validity and reliability of the empirical results.

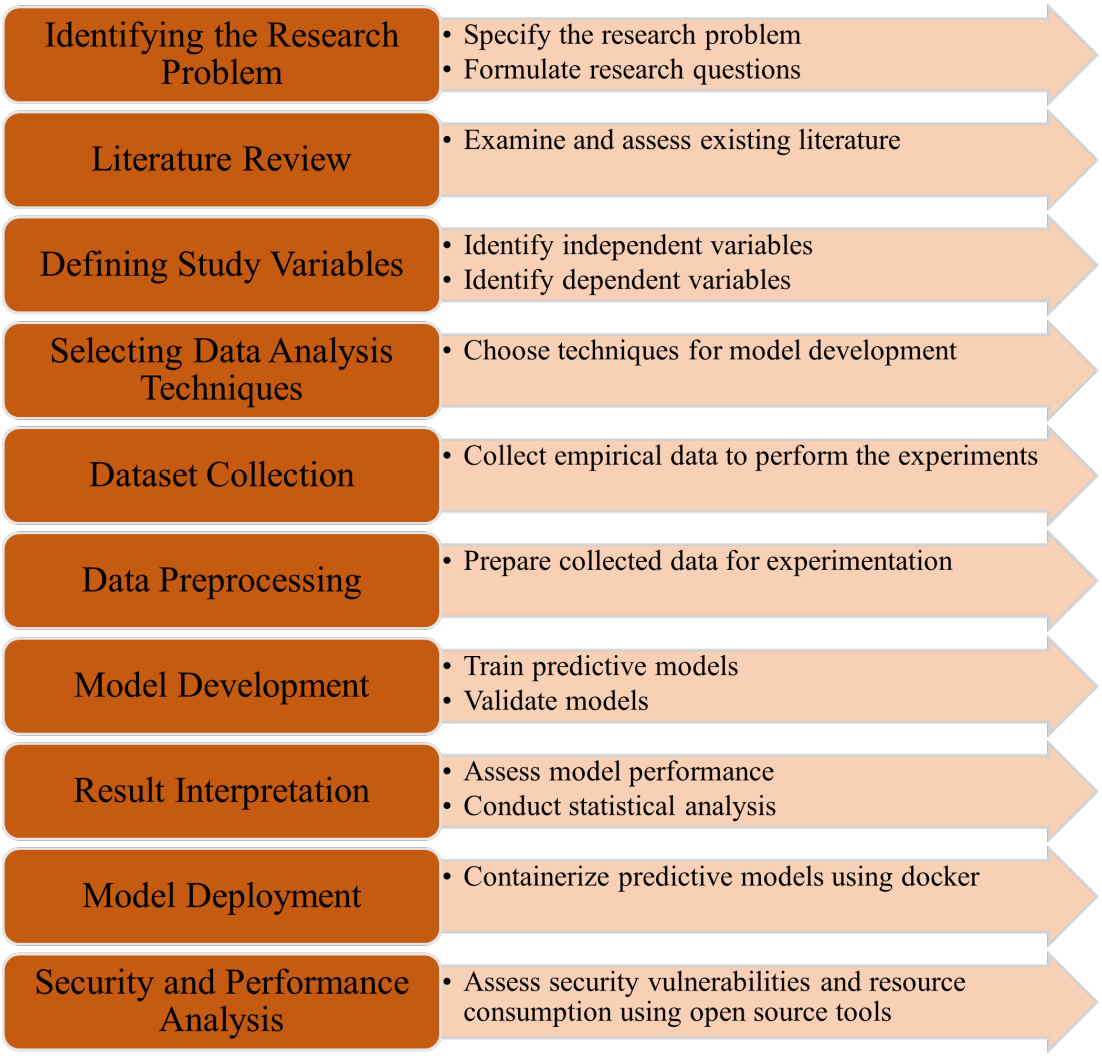



Figure 2.1: Research Methodology

## 2.3 Research Problem Identification

The first stage of the research process involves identifying and defining the research problem. At this stage, the central issue under investigation is carefully articulated and framed as specific research questions (RQs). These RQs provide clarity, focus, and direction for the study, serving as a foundation for designing experiments and investigations aimed at systematically addressing them. By formulating well-defined RQs, the research establishes a structured framework that guides the selection of data, tools, analytical methods, experimental design, and methodology. In this thesis, the following research questions are proposed, each aimed at deepening the understanding

of the problem under study and contributing to achieving the overall objectives of the research.

RQ1 (Systematic Literature Review Foundation): What is the current state of literature in the field of software container maintenance, and what gaps remains with respect to container image detection, container scheduling, container security, and container performance evaluation?

This question, addressed through the systematic literature review, provides the baseline understanding of the research landscape. The insights gained from RQ1 highlight persistent challenges and motivate the subsequent investigations of this thesis.

RQ2 (Predictive Accuracy and Robustness): How can DL frameworks be designed to improve the predictive accuracy and robustness of critical tasks such as SDP, cloud WF, and CFP?

RQ3 (Optimization and Federated Learning Paradigms): In what ways can metaheuristic optimization techniques and FL paradigms enhance the scalability, generalization, and privacy preservation of predictive models in software engineering and cloud environments?

RQ4 (Deployment, Security, and Performance Analysis): How can deployment-level analysis—specifically, resource consumption monitoring and vulnerability scanning—be systematically integrated into predictive modeling frameworks to ensure their practical applicability, security, and trustworthiness?

## 2.4 Literature Survey

To understand the scope of the research problem, a systematic literature survey on software container maintenance was conducted. Existing studies primarily focus on four aspects: container image detection, container scheduling, runtime security, and performance evaluation. Research shows that orchestration platforms such as Kubernetes, Docker Swarm, and Mesos are widely employed for scheduling and allocation, while security-oriented works emphasize vulnerability detection and mitigation using tools

like Anchore and Trivy. Performance evaluation studies rely on benchmarking and monitoring tools to analyze CPU, memory, disk, and network usage, ensuring container efficiency. The detailed process of conducting systematic literature review is described in chapter 3.

The survey also revealed important gaps, including the need for robust image tagging methods, advanced vulnerability assessment mechanisms, and performance objectives beyond resource utilization. These insights shaped the direction of this thesis by highlighting the necessity of developing predictive models that not only improve accuracy but also ensure scalability, security, and resource efficiency when deployed in containerized and distributed environments.

In line with these findings, the RQs of this thesis are framed to address both predictive modeling challenges (e.g., improving accuracy using DL, ensemble, and FL) and deployment-level challenges (e.g., resource consumption monitoring and vulnerability analysis). This dual focus ensures that the proposed solutions are not only theoretically strong but also practically viable and trustworthy in modern cloud and software engineering ecosystems.

## 2.5 Defining Study Variables

Variables represent characteristics, properties, or attributes that can change or take on different values within experimental or research contexts. In prediction models, these variables are categorized into independent and dependent variables, forming the fundamental structure for developing accurate predictive systems.

### 2.5.1 Independent Variables

Independent variables, also referred to as predictors, features, or explanatory variables, are the factors that researchers manipulate, observe, or control to examine their effects on the dependent variable. These variables serve as the input elements in predictive models and are not influenced by other variables within the study. They play a crucial

role in driving predictions and explaining or forecasting changes in outcomes. In this thesis, the independent variables vary depending on the application domain:

- For SDP, independent variables include static code metrics (e.g., lines of code, complexity, coupling) [122], and process metrics (e.g., code churn, developer activity) [184]. Table 2.1 describes static code metrics and process metrics used in SDP.

Table 2.1: Metrics Suite

| **Reference** | **Metric Name** | **Description** |
|---|---|---|
| [122] | Weighted Methods per Class (WMC) | It represents the total complexity of all methods. Since each method is assigned a complexity value of 1, it essentially corresponds to the number of methods in a class. |
| [122] | Depth of Inheritance Tree (DIT) | It denotes the highest level of the class within the inheritance hierarchy. |
| [122] | Number of Children (NOC) | It determines how many immediate child classes exist |
| [122] | Coupling Between Objects (CBO) | It represents how many classes are connected to a particular class. |
| [122] | Response for a Class (RFC) | It estimates how many methods of a specific class will respond when the class receives a given message. |
| [122] | Lack of Cohesion in Methods (LCOM) | It predicts how many methods within a specific class will execute in response to a given message. |
| [122] | Lack of Cohesion in Methods version 3(LCOM3) | It is calculated as: |
| [122] | Number of Public Methods (NPM) | It calculates how many public methods a class contains. |
| [122] | Data Access Metric (DAM) | This metric represents the ratio of private or protected attributes to the total number of attributes defined within a class |
| [122] | Measure of Aggregation (MOA) | This metric captures the degree of the part–whole relationship by examining attributes. It is measured as the number of class fields whose types are user-defined classes. |

| | | |
|---|---|---|
| [122] | Measure of Functional Abstraction (MFA) | This metric is defined as the ratio of the number of methods a class inherits to the total number of methods accessible to its member methods. |
| [122] | Efferent Coupling (Ce) | This metric indicates the number of classes on which the measured class depends. |
| [122] | Cohesion Among Methods (CAM) | This metric measures the degree of relatedness among the methods of a class based on their parameter lists |
| [122] | Inheritance Coupling (IC) | This metric quantifies the number of parent classes with which a given class is coupled. The coupling occurs when any inherited method from the parent class exhibits a functional dependency on the newly introduced or redefined methods in the subclass. |
| [122] | Coupling Between Methods (CAM) | The metric quantifies the total number of newly defined or redefined methods that are coupled with all inherited methods. |
| [122] | Average Method Complexity (AMC) | This metric calculates the average method size within each class, where the size of a method is defined as the number of Java bytecode instructions it contains. |
| | Afferent Coupling (Ca) | This metric indicates the number of classes that rely on the measured class. |
| [122] | Maximum McCabe's Complexity (Max (CC) | The highest CC value observed among the methods in the studied class |
| [122] | Average McCabe's Complexity (Avg (CC) | The average CC value of the classes under investigation. |
| [122] | Lines of Code (LOC) | This metric returns the number of lines in the java file. |
| [184] | Number of Revision (NR) | This metric measures how many times a Java class has been modified throughout the evolution of the examined software release. |
| [184] | Number of Distinct Committers (NDC) | This metric measures the number of developers who contributed changes to a Java class over the course of the software release's evolution. |
| [184] | Number of Modified Lines (NMC) | This metric measures the number of lines of source code that have been added to or removed from a Java class. |
| [184] | Number of Defects in Previous Version (NDPV) | This metric represents the number of defects resolved in a Java class over the course of the previous software release. |

- For CFP and WF, the independent variables are derived from resource utilization data, including CPU, memory, disk, and network usage. In the case of CFP, the focus is on predicting job and task failures using the GCT 2011 dataset. Specifically, the job event table contains features such as timestamp, missing information, job ID, event type, username, scheduling class, job name, and logical job name, while the task event table includes timestamp, missing information, job ID, task index (within the job), machine ID, event type, username, scheduling class, priority, resource requests (CPU, memory, disk space), and machine constraints. For constructing the job failure dataset, the job event table is combined with the task event table on the basis of the job ID, whereas the task failure dataset is derived directly from the task event table. In both cases, the event type serves as the dependent feature, while all remaining attributes act as independent features for modeling. Two important categorical features used in this study are scheduling class and priority. Each job or task is assigned a scheduling class that reflects its latency sensitivity, expressed as a numeric value from 0 to 3, where 0 corresponds to non-production tasks (e.g., development activities) and 3 represents highly latency-sensitive production tasks (e.g., handling user-facing, revenue-critical workloads). Similarly, priority is assigned as a small integer, where 0 indicates the lowest priority, and larger values correspond to higher importance. Tasks with higher priorities are preferentially allocated resources by the cluster scheduler. Specific ranges of priority values have special purposes:

  1. Free priorities – Tasks requesting resources with minimal internal charges.

  2. Production priorities – Tasks that are latency-sensitive and protected from eviction.

  3. Monitoring priorities – Tasks assigned to monitor and track the performance of lower-priority tasks.

  Finally, resource requests define the maximum CPU, memory, or disk space that a task is allowed to consume, and therefore directly influence scheduling, performance, and the likelihood of job or task termination.

- For HAR, the independent variables are the x, y, and z coordinate signals collected through embedded sensors.

By leveraging these diverse sets of predictors across domains, the thesis develops robust models tailored to the specific challenges of software engineering, cloud reliability, and sensor-based activity recognition.

### 2.5.2 Dependent Variables

Dependent variables, conversely, represent the outcomes or results being measured and predicted. These variables are called ”dependent” because their values rely entirely on the independent variables included in the model. In prediction models, the dependent variable is what the system attempts to forecast based on the patterns learned from independent variables. In the context of this thesis, the dependent variables are defined according to the application domains as follows:

- For SDP, the dependent variable is the fault label of a software module, indicating whether a module is fault-prone or non-fault-prone.
- For CFP, the dependent variable is the event type attribute in the job and task failure datasets. This attribute indicates the final termination status of a job or task in the cloud data center. Table 2.2 describes the details of type of event in selected dataset. Specifically, the event type is categorized into two classes: successful (event code = 4, representing FINISH) and failure (all other event codes, such as EVICT, FAIL, KILL, LOST). Thus, the dependent variable serves as the binary indicator that distinguishes whether a job or task completed normally or encountered a failure during its lifecycle.
- For WF, the dependent variable corresponds to the future workload demand (e.g., CPU usage, memory usage), measured as a continuous value over time.
- For HAR, the dependent variable is the activity class label, which represents the type of activity (e.g., walking, sitting, standing, running) as inferred from the

sensor signals.

Table 2.2: Event Types

| **Event Codes** | **Event Name** | **Description** |
|---|---|---|
| 0 | Submit | A job or task becomes eligible to be scheduled. |
| 1 | Schedule | A job or task is assigned to a machine. (Execution may not begin immediately due to factors like code transfer. For jobs, this is logged the first time any of its tasks is scheduled.) |
| 2 | Evict | A job or task is removed from a machine because of reasons such as a higher-priority task taking precedence, overcommitment of resources, machine failure or shutdown, or loss of disk data. |
| 3 | Fail | A job or task is removed from scheduling (or rarely, becomes ineligible while waiting) due to a task failure. |
| 4 | Finish | A job or task successfully completes execution. |
| 5 | Kill | A job or task is cancelled, either by the user, a driver program, or because another dependent job or task failed. |
| 6 | Lost | A job or task is presumed terminated, but no termination record exists in the source data. |
| 7 | Update_Pending | While waiting for scheduling, a job or task has its resource needs, scheduling class, or constraints modified. |
| 8 | Update_Running | While already scheduled, a job or task has its resource needs, scheduling class, or constraints modified. |

Thus, the dependent variables serve as the targets that the proposed predictive models aim to estimate. They enable effective fault detection across software systems and improve workload optimization in cloud environments. These variables also support enhanced reliability by predicting failures in advance. Furthermore, they contribute to efficient resource utilization and system performance. Overall, they facilitate accurate activity recognition and robust decision-making.

## 2.6 Data Analysis Techniques

In this thesis, a combination of ML, ensemble learning, DL, and FL techniques have been employed to design robust predictive models for diverse application domains including SDP, WF, CFP, and HAR.

ML techniques encompass traditional algorithms designed to capture and model the underlying relationships between predictor variables (features) and the target variable (outcome). ML techniques included Decision Tree (DT), k-Nearest Neighbors (KNN), Logistic Regression (LR), Naïve Bayes (NB), and Support Vector Machine (SVM). These algorithms served as baseline models, leveraging mathematical and statistical formulations to learn from historical data and establish fundamental predictive capabilities.

Ensemble techniques leverage the combined strength of multiple models to overcome the limitations of individual learners and improve prediction accuracy, robustness, and generalization. Methods such as Random Forest (RF), Gradient Boosting (GB), and Adaptive Boosting (AB) construct collections of models and aggregate their outputs, thereby reducing variance and minimizing the risk of overfitting. In this thesis, ensemble approaches—particularly RF and Stacking were employed to achieve more stable and reliable predictions across diverse tasks. By integrating the complementary strengths of different learners, ensemble methods enhanced overall system performance, yielding predictive models that are more resilient and effective in real-world deployment scenarios.

DL techniques in this thesis include architectures such as Convolutional Neural Networks (CNN), Long Short-Term Memory (LSTM) networks, Bidirectional LSTM (BiLSTM), and hybrid models like CNN–LSTM and CNN–BiLSTM. These models are designed to capture intricate and non-linear patterns in the data by leveraging multilayer neural networks. CNNs are particularly effective in extracting spatial and hierarchical features, while LSTMs and BiLSTMs are suited for modeling temporal dependencies in sequential data. The hybrid models combine the strengths of convolutional fea-

ture extraction and recurrent sequence learning, thereby improving performance on complex, high-dimensional datasets such as workload traces, failure logs, and sensor signals. Unlike traditional ML techniques, DL models have a greater capacity to represent complex, high-dimensional relationships, making them well suited for tasks requiring high-level feature extraction and temporal pattern recognition. By learning through deep layered architectures, these models can generalize effectively to unseen data, delivering powerful predictions across diverse application domains.

FL was incorporated to address challenges of privacy and scalability in distributed environments, particularly for WF and SDP. By enabling collaborative model training across multiple clients without sharing raw data, FL ensured data confidentiality while achieving high predictive accuracy.

To maximize the predictive performance of the above models, several HPT approaches were employed. These included:

- Manual Tuning, where parameters were adjusted through iterative experimentation.
- Optuna, an efficient automated optimization framework leveraging Bayesian optimization and pruning strategies for effective hyperparameter search.
- GWO and WOA, which are population-based metaheuristics inspired by natural behaviors and used to optimize deep learning models.
- Hybrid GWO–WOA, combining the exploration capabilities of GWO with the exploitation strength of WOA, yielding superior convergence and accuracy.

  These optimization strategies significantly enhanced the performance of predictive models, ensuring better generalization, stability, and computational efficiency across diverse tasks.

  By systematically integrating ML, ensemble, DL, FL, and optimization techniques, this thesis contributes predictive models that are not only accurate and scalable but also secure, resource-aware, and practical for real-world deployment in software engineering and cloud environments.

### 2.6.1 Decision Tree

A Decision Tree is a supervised learning algorithm that recursively splits the data into smaller subsets based on feature values, ultimately forming a tree-like model of decisions. Each internal node represents a test on an attribute, each branch represents an outcome of the test, and each leaf node corresponds to a class label. The model learns by selecting features that provide the maximum discriminatory power between classes [335].

The splitting criterion is often based on Information Gain or Gini Impurity. For example, the Gini Index is computed as:

$$Gini(D) = 1 - \sum_{i=1}^{c} p_i^2 \tag{2.1}$$

where is the proportion of samples belonging to class i in dataset D and c is the number of classes.

A lower Gini value indicates purer nodes. Decision Trees are intuitive and interpretable, making them suitable for software defect prediction and failure classification. However, they are prone to overfitting if not properly pruned.

### 2.6.2 K Nearest Neighbor

KNN is an instance-based, non-parametric learning algorithm. Instead of building an explicit model, it stores all training instances and classifies new data points based on their similarity to the stored samples. A query instance is classified by finding the majority class among its k nearest neighbors in the feature space. The most common distance metric is the Euclidean distance [62][222]. It can be computed with the following mathematical formula:

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{2.2}$$

Where x and y are two data points with n features.

KNN is simple and effective, especially when the decision boundary is irregular. Its performance, however, depends on the choice of k, the distance metric, and normalization of features.

### 2.6.3 Logistic Regression

LR is a statistical model used for binary classification that predicts the probability of an outcome belonging to a particular class. Instead of fitting a straight line, it applies the logistic (sigmoid) function to map predictions into the range $\{0, 1\}$. The model computes the probability of class y=1 given features x as:

$$P(y = 1 \mid x) = \frac{1}{1 + e^{-(w^T x + b)}} \tag{2.3}$$

Where w is the coefficient vector and b is the intercept.

A threshold (commonly 0.5) is applied to convert the probability into a class label. LR is easy to implement, interpretable, and works well when the relationship between features and outcome is approximately linear. However, it may underperform in complex, non-linear classification tasks compared to DL approaches. A detailed description of LR is given by Basili *et al.*(1996) [26] and Hosmer and Lemeshow(1989) [102].

### 2.6.4 Naïve Bayes

NB is a probabilistic classifier grounded in Bayes' Theorem, which calculates the posterior probability of a class given a set of features. It assumes conditional independence between features, making computation efficient even with high-dimensional data [244]. The classification rule is defined as:

$$P(C_k \mid x_1, x_2, \ldots, x_n) = \frac{P(C_k) \prod_{i=1}^{n} P(x_i \mid C_k)}{P(x_1, x_2, \ldots, x_n)} \tag{2.4}$$

Where $C_k$ is the class and $x_1, x_2, \ldots, x_n$ are the features

The model predicts the class with the highest posterior probability. Despite its "naïve" independence assumption, NB often performs competitively in text classification, bug prediction, and defect-prone module identification.

### 2.6.5 Support Vector Machine

SVM is a powerful supervised learning algorithm widely used for classification and regression tasks. Its primary goal is to find the optimal hyperplane that best separates data points of different classes while maximizing the margin, i.e., the distance between the hyperplane and the nearest data points of each class, known as support vectors. This margin maximization makes SVM robust and less prone to overfitting, particularly in high-dimensional feature spaces [187]. SVM supports various kernel functions, which enable it to adapt to different types of data distributions by transforming the input space:

- Linear Kernel: Applies a linear transformation and is most effective when the data is already linearly separable. Mathematically, it can be formulated as:

$$K(x, x') = x^T x' \tag{2.5}$$

- Polynomial Kernel: Maps input features into a higher-dimensional polynomial space, thereby allowing SVM to capture polynomial relationships between features. Mathematically, it can be computed as:

$$K(x, x') = (x^T x' + c)^d \tag{2.6}$$

  where c is a constant that trades off the influence of higher-order terms, and d is the degree of the polynomial.

- Radial Basis Function (RBF) Kernel: The most widely used kernel in practice, as it maps the data into an infinite-dimensional space, enabling SVM to construct

highly flexible and non-linear decision boundaries.

- Sigmoid Kernel: Functions in a manner similar to the activation function of a neural network, making it suitable for specialized cases where such transformations are beneficial.

SVM is particularly effective in tasks involving high-dimensional data, sparse datasets, and cases with clear margins of separation. However, its computational complexity can be high for very large datasets, and performance depends heavily on careful selection of kernel functions and hyperparameters such as C (regularization) and $\gamma$(kernel coefficient).

### 2.6.6 Random Forest

RF is an ensemble learning technique that builds multiple DTs during training and aggregates their predictions to improve accuracy and robustness. It combines the concepts of bagging (bootstrap aggregating) and feature randomness, which collectively reduce variance and minimize the risk of overfitting compared to individual DTs [277][279]. The algorithm works by:

- Randomly selecting subsets of the training dataset with replacement (bootstrap samples).
- Constructing a DT for each sample by selecting a random subset of features at each split.
- Aggregating predictions from all trees using majority voting for classification tasks or averaging for regression tasks.

The prediction for classification is expressed as:

$$y' = \text{mode}\{h_1(x), h_2(x), \ldots, h_T(x)\} \tag{2.7}$$

Where $h_t(x)$ represents the prediction of the t$^{th}$ DT and T is the total number of trees.

RF is particularly effective because it reduces variance while maintaining low bias, making it well-suited for handling large, high-dimensional, and noisy datasets. In this thesis, RF was used as a baseline ensemble method for defect prediction and failure classification, offering interpretability and strong generalization performance.

### 2.6.7 Stacking

Stacking, or stacked generalization, is an advanced ensemble learning technique that combines multiple base classifiers with a meta-classifier to achieve superior predictive performance. Unlike bagging and boosting, stacking leverages the diversity of heterogeneous models rather than focusing solely on variance reduction or error reweighting [69][199]. The process of stacking involves two levels:

Level-0 (Base Learners): Multiple classifiers such as CNN, LSTM, BiLSTM, RF, LR, and SVM are trained independently on the same dataset. Each model provides its predictions for the training data.

Level-1 (Meta Learner): A meta-classifier (commonly LR or a simple ANN) is trained on the predictions of the base learners to generate the final output. Mathematically, if the base learners are $h_1(x)$, $h_2(x)$, ….., $h_m(x)$, then the meta-classifier H learns a mapping :

$$H(x) = g(h_1(x), h_2(x), ....., h_m(x))$$

Where g() is the function learned by the meta-learner.

Stacking capitalizes on the strengths of diverse models, mitigating their individual weaknesses, thereby achieving higher accuracy and robustness in prediction tasks.

### 2.6.8 Artificial Neural Network (ANN)

ANNs are the most fundamental class of DL models, inspired by the biological structure of the human brain. They consist of interconnected units known as artificial neurons, organized into an input layer, one or more hidden layers, and an output layer. Each neuron receives inputs, multiplies them by associated weights, adds a bias term,

and applies a non-linear activation function (such as sigmoid, tanh, or ReLU) to determine its output. Each neuron in hidden layer computes:

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

where $x_i$ denotes input features, $w_i$ denotes weights, b represents bias, and f() represents activation function (Relu, sigmoid, tanh, etc.).

The full network computes:

$$f_\theta(\vec{x}) = \text{output}$$

and is trained to minimize an error function, e.g., mean squared error (MSE):

$$E(X, \theta) = \frac{1}{2} \sum_i \left(y_i - f_\theta(\vec{x}_i)\right)^2$$

Where $\theta$ represents all parameters (weights and biases).

ANNs learn by adjusting the weights and biases through an optimization algorithm (such as gradient descent) that minimizes the error between predicted and actual outputs. Backpropagation is used to compute gradients and update parameters across layers. Although ANNs are relatively simple compared to more advanced DL architectures, they provide a flexible framework capable of approximating complex, non-linear relationships. Their strength lies in generalization across diverse datasets, but they can struggle with sequential or high-dimensional data.

### 2.6.9 Convolutional Neural Network (CNN)

CNNs are specialized DL architectures designed to process data with spatial or grid-like topology, such as images, time-series signals, or structured logs. A CNN is composed of three main types of layers: convolutional layers, pooling layers, and fully connected layers [63]. The convolutional layers apply learnable filters (kernels) to input data, performing convolution operations that slide over the input and capture local features. Pooling layers (e.g., max-pooling, average-pooling) reduce feature map di-

mensions by selecting the most important features, improving computational efficiency and reducing overfitting. Finally, fully connected layers aggregate extracted features and perform classification or regression. A CNN processes spatial data using convolutions [133]. The mathematical operation for a convolution layer to compute a single output pixel is:

$$S(i,j) = \sum_{m=1}^{M} \sum_{n=1}^{N} I(i+m-1,\, j+n-1) \cdot K(m,n)$$

Where I is the input image, K is the kernel(filter), and S(i, j) is the resulting feature map at position (i, j). For dense layers, mathematical formula is:

$$z = Wx + b$$

Where W are weights, x is the input vector, and b is bias. CNNs are highly effective in extracting spatial features and noise-robust patterns. Their strength lies in automatic feature extraction, eliminating the need for extensive manual feature engineering.

### 2.6.10 Long Short-Term Memory (LSTM)

LSTM networks are an advanced type of Recurrent Neural Network (RNN) designed to address the limitations of traditional RNNs in learning long-term dependencies due to vanishing and exploding gradients. LSTMs incorporate memory cells and gating mechanisms that regulate information flow [296]. These gates include forget gate, input gate, and output gate.

- Forget gate decides which past information to discard. Mathematically, it can be expressed as:

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

  Where $W_f$ represents the weight matrix, $[h_{t-1}, x_t]$ represents the concatenation of the previous cell output ($h_{t-1}$) and the current input ($x_t$), $b_f$ is the bias with the forget gate, $f_t$ is the forget gate activation, and $\sigma$ is the sigmoid activation

function.

- Input gate decides how much new information to store. The operations involved are represented by the following equations:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)\,, \quad \tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

Where $i_t$ is the input gate activation, $\tilde{C}_t$ is the candidate cell state vector, $W_i$ and $W_c$ denote the weight matrices for the input gate, and $b_i$ and $b_c$ are the bias values. Furthermore, the cell state is updated using the following equation:

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t$$

Where $C_t$ is the updated state and $C_{t-1}$ is the previous state.

- The output gate controls how much information from the cell state flows into the hidden state. The operations involved are represented by the following equations:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o), \quad h_t = o_t \cdot \tanh(C_t)$$

Where $o_t$ is the output gate activation, $h_t$ is the hidden state, $W_o$ is the output gate weight matrix, and $b_o$ is the bias value.

The internal cell state preserves long-term contextual information, allowing LSTMs to effectively model sequential and time-series data. The ability of LSTMs to retain long-term dependencies significantly improved model accuracy compared to simpler ANNs and CNNs [80].

### 2.6.11 Bi-Directional LSTM (BiLSTM)

BiLSTMs extend the standard LSTM architecture by processing input sequences in both forward ($\overrightarrow{h_t}$) and backward ($\overleftarrow{h_t}$) directions. While a traditional LSTM only learns dependencies from past to future, BiLSTM captures contextual information from both

directions, leading to more comprehensive temporal modeling [100]. The hidden state at each time step is formed by concatenating forward and backward outputs:

$$h_t = [\overrightarrow{h_t}; \overleftarrow{h_t}]$$

This dual representation enables BiLSTMs to account for both past and future context when making predictions. While BiLSTMs are more computationally intensive than LSTMs, they achieve superior predictive performance.

### 2.6.12 CNN-LSTM

The CNN-LSTM is a hybrid DL architecture that combines the strengths of convolutional and recurrent models. The CNN layers are employed first to extract spatial or local features from the raw input data, which are then passed to LSTM layers for temporal modeling. This combination is highly effective for data with spatio-temporal properties. For instance, CNN layers identify local usage spikes or sensor correlations, while LSTMs model how these patterns evolve over time [216][309]. By integrating both spatial and temporal learning, CNN-LSTM outperformed standalone CNNs or LSTMs in capturing complex dynamics.

### 2.6.13 CNNBiLSTM

CNN-BiLSTM extends the CNN-LSTM architecture by replacing LSTM layers with bidirectional LSTM layers. This modification allows the model not only to capture spatial features through CNN but also to exploit temporal dependencies in both forward and backward directions. The CNN-BiLSTM thus combines feature extraction with comprehensive bidirectional sequence modeling [39]. The trade-off, however, is increased computational cost and model complexity. Despite this, CNN-BiLSTM achieved the highest predictive accuracy across multiple tasks, underscoring the importance of hybrid DL approaches for complex real-world datasets.

### 2.6.14 Federated Learning (FL)

Centralized learning is the conventional method for developing ML and DL models, where data is gathered and stored on a server, the model is trained on that server, and predictions are also made there. This approach can pose privacy risks to user data stored on the server. To address this, Google introduced FL, a decentralized approach that enhances privacy by keeping data on local client devices or nodes while training the model directly on these devices. Figure 2.2 shows the difference between conventional approach and FL approach. FL is a form of distributed ML designed to protect data privacy. In FL systems, a central server manages the coordination of model training across multiple distributed client devices where the data resides. The training occurs locally on the client devices, ensuring that data does not need to be transferred outside of these devices.

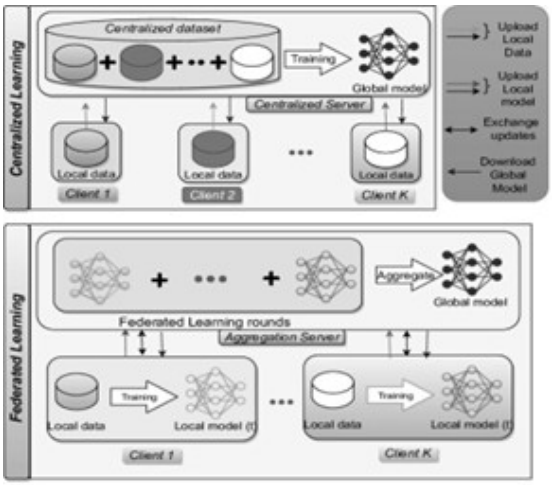



Figure 2.2: Traditional vs. FL

Privacy of data and efficient communication are two major factor to adopt FL approach. This makes FL suitable for large-scale, privacy-sensitive applications. Zhang *et al.* [325] and Li *et al.* [160] reviewed all the existing studies related to FL. They discussed the challenges, future directions, and applications of FL. It has been used in wireless communication [228][282][20][19], healthcare systems [163][174][153][192], and mobile devices for recommending the services [11][76][297][240].

## 2.7 Empirical Data Collection

Empirical data collection forms the foundation of this thesis, as the effectiveness of the proposed predictive frameworks depends heavily on the availability of reliable and representative datasets. Since the research focuses on SFP, cloud WF, CFP, and HAR, it was essential to select datasets that not only reflect real-world complexities but also serve as established benchmarks for fair comparison with existing approaches.

The datasets chosen in this thesis fulfill three major objectives. First, they provide benchmark status, ensuring that the results can be directly compared with prior studies and validated by the broader research community. For instance, PROMISE datasets are widely adopted for SFP, while Google and Alibaba cluster traces are recognized standards in CC research. Second, they capture real-world relevance, as the data originates from operational systems such as open-source software projects, large-scale cloud clusters, and sensor-based applications. This ensures that the models are validated under conditions similar to practical deployment. Finally, they offer diversity and generalization, covering multiple modalities, static code and process metrics, cloud resource utilization traces, and sensor signals, thereby enabling the evaluation of proposed models across different domains and contexts. By including datasets from software engineering, CC, and HAR applications, this thesis demonstrates the scalability, adaptability, and robustness of the proposed frameworks. The following subsection provides a detailed description of datasets used in this thesis.

### Dataset Details

To operationalize the empirical investigations, four categories of datasets were employed, corresponding to the major application domains of the thesis: SDP, CFP, WF, and HAR. Each dataset was selected based on its benchmark status, richness of features, and suitability for predictive modelling.

- **Software Defect Prediction**

  We have collected the dataset from a publicly available metric repository main-

tained by Madeyski and Jureczko [184]. We chose datasets developed using the Java language. These datasets consist of both static code metrics such as WMC, LOC, LCOM, NOC, DIT, etc., and process metrics such as NR, NML, NDC, and NDPV as independent variables. Each module is labeled as defective or non-defective, enabling supervised learning and facilitating comparisons with prior SDP research.

- **Cloud Failure Prediction**

  GCT v2011 [2] dataset was used for predicting task and job failures in cloud environments. This large-scale dataset records millions of events with detailed resource utilization metrics (CPU, memory, disk, and network usage), task/job execution logs, and associated failure statuses. Its scale and diversity make it particularly suitable for training DL and FL models for proactive failure management.

- **Workload Forecasting**

  Both GCT (v2011 and v2019) and Alibaba Cluster Trace (v2018 and v2020) datasets were adopted for WF tasks. These datasets provide time-series workload records collected from real-world production clusters, capturing fluctuations in CPU, memory, disk, and network demand. The inclusion of two different providers ensures that the forecasting models are not biased toward a single environment and can demonstrate cross-cloud generalizability.

- **Human Activity Recognition**

  To evaluate the adaptability of predictive models, three benchmark HAR datasets were employed:

  1. **WISDM Dataset:** Sensor data collected from smartphones and smartwatches, including accelerometer and gyroscope readings, covering activities such as walking, jogging, climbing stairs, and sitting.
  2. **UCI HAR Dataset:** Data collected from 30 volunteers performing daily activities while wearing a smartphone equipped with embedded sensors, providing x, y, z axis readings of accelerometer and gyroscope signals.

3. **FLAAP Dataset:** A fine-grained activity recognition dataset offering diverse sensor readings across multiple participants, designed to support detailed classification of human activities.

The selection of these datasets ensures that the proposed models are validated against heterogeneous, large-scale, and real-world scenarios, covering the spectrum of software systems, cloud infrastructures, and IoT-based applications. In the subsequent subsection, we discuss the data preprocessing and imbalance handling techniques applied to these datasets, which form a crucial step in preparing the data for DL, optimization, and federated modelling techniques.

## 2.8 Data Preprocessing

Data preprocessing is a critical step in preparing datasets for predictive modelling, as raw data often contains inconsistencies, redundancies, or imbalances that can significantly affect model performance. The process began with data cleaning, where redundant entries such as duplicate rows, irrelevant columns, and inconsistent labels were removed. In software datasets, duplicate modules and redundant metrics were discarded, while in cloud traces, incomplete or corrupted job/task records were excluded. Missing values were then handled either by imputation techniques, such as mean, median, or mode replacement, or by removing records with excessively high levels of missingness to minimize bias. To address the issue of varying scales across features, normalization and scaling were applied—using either Min-Max normalization, which scales values between 0 and 1, or standard scaling, which centers data around zero with unit variance. This step was particularly important for cloud resource metrics (CPU, memory, disk, GPU usage) to ensure comparability across traces. Additionally, many datasets exhibited class imbalance, such as a higher proportion of non-defective modules compared to defect-prone modules, or more successful tasks compared to failed tasks in cloud traces. To mitigate this, balancing techniques such as SMOTE were employed, ensuring fair representation across classes and preventing biased predictions. Furthermore, domain-specific preprocessing was applied—for example, extracting statistical

features from HAR sensor streams, smoothing and windowing time-series traces for WF, and aligning task/job records with resource usage data in CFP. These steps collectively ensured that the datasets were clean, balanced, and feature-rich, providing a reliable foundation for training the proposed predictive models.

### Data Normalization

Data normalization is a crucial step in data preprocessing across all the domains addressed in this thesis, as it ensures that features are represented on a comparable scale, thereby enhancing both model performance and accuracy [75]. The primary objective of normalization is to prevent attributes with larger numerical ranges from dominating the learning process, while also improving the convergence rate of algorithms and mitigating the influence of outliers [16] [223]. Commonly used techniques include Min-Max scaling, which rescales values to a $[0, 1]$ range, and Z-score normalization, which standardizes features to have a mean of zero and unit variance. The choice of normalization technique should be guided by the data distribution and must be applied consistently across training and testing datasets. The benefits of normalization extend beyond computational efficiency. Properly normalized data yields improved model accuracy, faster training, and more meaningful feature comparisons, enabling predictive models to generalize more effectively across diverse datasets.

**Min-Max Scaling** It transforms feature values into a fixed range, typically $[0, 1]$, preserving the relative relationships among features while making them directly comparable [133]. The formula is:

$$X_{\text{scaled}} = \frac{X - X_{\min}}{X_{\max} - X_{\min}} \tag{2.8}$$

Where $X$ is the original feature value, $X_{\min}$ and $X_{\max}$ are the minimum and maximum values of feature $X$, and $X_{\text{scaled}}$ is the normalized value.

**Z-score Normalization (Standard Scaling)** It transforms features by centering them at zero mean with a unit variance, thereby reducing the effect of varying feature distributions [222]. The formula is:

$$X_{\text{scaled}} = \frac{X - \mu}{\sigma} \tag{2.9}$$

Where $X$ is the original feature value, $\mu$ is the mean of the feature, $\sigma$ is the standard deviation of the feature, and $X_{\text{scaled}}$ is the normalized value.

## 2.9 Data Imbalance Handling

Imbalanced class distributions are a prevalent challenge in predictive modeling, particularly when minority class instances are significantly underrepresented and may be of greatest practical interest (e.g., defect-prone modules, rare failures, or minority activities). To address this, the Synthetic Minority Oversampling Technique (SMOTE) was employed as an oversampling strategy. SMOTE generates synthetic minority class samples by interpolating between existing minority samples within their feature space neighborhoods, rather than replicating original instances. Specifically, for each minority instance, *k*-nearest neighbors are identified, and new synthetic samples are created by random linear combinations of the feature values between the original instance and its neighbors. This approach results in a more balanced training set, reducing classifier bias toward the majority class and improving the detection of minority class instances.

By integrating SMOTE within the data preprocessing workflow, this research ensures that predictive models are trained on balanced datasets, enabling more robust, accurate, and generalizable results across all studied domains. [33]

## 2.10 Development and Validation of Predictive Models

The predictive models developed in this thesis span four key domains: SDP, CFP, WF, and HAR. The models in this thesis are developed using supervised ML and DL techniques, supported by a range of data analysis methods (as discussed in Section 2.6). The training datasets consist of both the independent variables and the dependent variables. The data analysis technique enables the model to learn the mapping between

independent and dependent variables. Once the model is constructed, it undergoes validation to assess its predictive performance. In this process, only the independent variables are provided to the trained model, which then predicts the values of the dependent variable (predicted label). The predicted labels are compared with the actual labels to determine the accuracy and effectiveness of the model. Although several validation methods exist, such as leave-one-out, cross-validation, and bootstrapping [273], this study employs the train–test split method due to its simplicity and efficiency. In this approach, the dataset is divided into two disjoint subsets: one used for training the model and the other for testing. For all experiments, the datasets were divided into two disjoint subsets: 80% for training and 20% for testing. This ratio strikes a balance by ensuring that the models are trained on a sufficiently large portion of the data while preserving a meaningful share for unbiased performance evaluation. By evaluating the models on previously unseen test data, the approach provides a reliable estimate of generalization capability across all domains.

This systematic process of development and validation ensures that the predictive models presented in this thesis are not only accurate and robust in experimental settings but also scalable and deployable in real-world environments such as software projects, cloud data centers, and IoT-based HAR systems.

## 2.11 Performance Evaluation Metrics

In this thesis, performance metrics serve as the primary means of evaluating the effectiveness and reliability of the proposed predictive models across multiple domains, including SDP, CFP, WF, and HAR. These metrics provide quantitative measures that assess how accurately a model captures the underlying patterns in the data and how well it generalizes to unseen instances. The choice of evaluation metrics is critical, as they offer insights into different aspects of model performance such as overall accuracy, error rates, discriminative ability, robustness against imbalanced data, and reliability of predictions in real-world scenarios. To capture these dimensions, the thesis employs a diverse set of metrics tailored to classification and regression tasks:

**For classification problems** (SDP, CFP, HAR), metrics such as the confusion matrix, accuracy, precision, recall, F1-score, AUC, and MCC are used. These metrics collectively provide a balanced view of model effectiveness, from simple correctness (accuracy) to performance under class imbalance (MCC, F1-score).

**For regression tasks** (WF), metrics such as Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), Mean Absolute Percentage Error (MAPE), and the Coefficient of Determination ($R^2$) are employed. These capture prediction error magnitude, deviation from actual workload trends, and the explanatory power of the model.

### Confusion Matrix

The confusion matrix is a tabular representation of classification outcomes, consisting of True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN):

- **TP (True Positive):** Correctly predicted positive class (e.g., defective or failed task/job)
- **TN (True Negative):** Correctly predicted negative class (e.g., non-defective or successful task/job)
- **FP (False Positive):** Incorrectly predicted positive class (e.g., non-defective predicted as defective)
- **FN (False Negative):** Incorrectly predicted negative class (e.g., defective predicted as non-defective)

Table 2.3: Confusion Matrix Representation

| **Actual / Predicted** | **Predicted Positive** | **Predicted Negative** |
|---|---|---|
| **Actual Positive** | True Positive (TP) | False Negative (FN) |
| **Actual Negative** | False Positive (FP) | True Negative (TN) |

**Accuracy:** Accuracy refers to how correctly the model classifies the class of an instance. It is calculated as ratio of correctly predicted instances to total number of instances. Equation 2.10 represents the mathematical formula for calculating the accuracy.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.10}$$

**Precision:** Precision is defined as the ratio of the number of TP outcomes to the total number of TP and FP outcomes. The precision score indicates how well the model can detect negative instances. Equation 2.11 represents the mathematical formula for calculating precision.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2.11}$$

**Recall (Sensitivity):** Recall (Sensitivity) is measured by taking the ratio of the number of TP outcomes to the total number of TP and FN outcomes. The recall score indicates how well the model can detect positive instances. Equation 2.12 represents the mathematical formula for calculating recall.

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2.12}$$

**F1 Score:** F1 score is the average of precision and recall, which indicate the model's capacity to identify negative samples and positive samples, respectively. The classification model is more robust if the F1 score is higher. Equation 2.13 represent the mathematical formula for calculating f1 score.

$$\text{F1 Score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.13}$$

**AUC (Area Under the ROC Curve):** The ROC curve plots the True Positive Rate (TPR) against the False Positive Rate (FPR) at various thresholds. AUC is represented as the curve between 1-specificity (FPR) and sensitivity (TPR) [66][91]. The value of AUC lies between 0 and 1. Equation 2.14 represents the mathematical formula for calculating TPR and FPR and equation 2.15 represents the mathematical formula for calculating AUC.

$$\text{TPR} = \frac{TP}{TP + FN}, \quad \text{FPR} = \frac{FP}{FP + TN} \tag{2.14}$$

$$\text{AUC} = \int_0^1 TPR(FPR)\, d(FPR) \tag{2.15}$$

**Matthews Correlation Coefficient (MCC):** MCC is the correlation coefficient between actual outcome and predicted outcome of an instance [91]. The value of MCC lies between -1 and 1. Equation 2.16 represents the mathematical formula for calculating MCC.

$$\text{MCC} = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{2.16}$$

**Mean Absolute Error (MAE):** It measures the average magnitude of errors between predicted and actual values, without considering direction. Equation 2.17 represents the mathematical formula for calculating MAE.

$$\text{MAE}(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i| \tag{2.17}$$

**Mean Squared Error (MSE):** MSE computes the average of the squared differences between actual and predicted values. By squaring errors, it penalizes larger deviations more heavily than smaller ones. Equation 2.18 represents the mathematical formula for calculating MSE.

$$\text{MSE}(y, \hat{y}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.18}$$

**Root Mean Squared Error (RMSE):** The square root of MSE, which expresses the error in the same units as the original data. It is more interpretable than MSE since it directly represents the scale of prediction errors. Equation 2.19 represents the mathematical formula for calculating RMSE.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{2.19}$$

**Mean Absolute Percentage Error (MAPE):** Expresses errors as percentages, providing a scale-independent measure of accuracy. Equation 2.20 represents the mathematical formula for calculating MAPE.

$$\text{MAPE}(y, \hat{y}) = \frac{100\%}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \tag{2.20}$$

**Coefficient of Determination ($R^2$):** It represents the proportion of variance in the actual data explained by the model. Equation 2.21 represents the mathematical formula for calculating $R^2$.

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{2.21}$$

Where:

- $y_i$: actual value
- $\hat{y}_i$: predicted value
- $\bar{y}$: mean of actual values

## 2.12 Statistical Test

This thesis employs the Friedman statistical test followed by the Nemenyi post hoc test to evaluate the statistical significance of performance differences among multiple predictive techniques across multiple datasets.

### Friedman Test

The Friedman test is a non-parametric statistical test used to compare the performance of multiple models over several datasets. It ranks the models for each dataset individually and then analyzes the average ranks to determine if there are statistically significant differences in performance [187][67].

The hypotheses are defined as:

- **Null Hypothesis ($H_0$):** There exists no significant difference between the performance of different techniques.
- **Alternative Hypothesis ($H_a$):** There exists a significant difference between the performance of the different techniques.

**Steps to conduct the Friedman test:**

1. Organize and sort the performance values of all the techniques for a specific dataset in descending order. Allocate ranks from 1 to k where rank 1 is assigned to the best-performing technique and rank k to the worst. In case of ties, assign the average of the ranks that would have been assigned to those observations.
2. For each technique, compute the sum of ranks allocated across all datasets. The total rank for each technique is represented by $R_1, R_2, \ldots, R_k$.
3. Finally, the friedman test statistics is computed by the following mathematical equation 2.22:

$$\chi_F^2 = \frac{12n}{k(k+1)} \sum_{i=1}^{k} R_i^2 - 3n(k+1) \tag{2.22}$$

Where:

- $n$ = number of datasets
- $k$ = number of techniques
- $R_i$ = total rank of the $i^{th}$ technique

4. 1. To justify the null hypothesis, we compare the calculated value of friedman statistics with the chi square tabulated value at degree of freedom (df) k-1 and chosen significance value (e.g. $\alpha$ = 0.05).

5. If $\chi^2_{\text{calculated}} > \chi^2_{\text{tabulated}}$, then null hypothesis is rejected, we can conclude that there exists significant difference between the performance of the techniques. If null hypothesis gets rejected, we apply nemenyi post hoc test to analyze the performance of which technique differ significantly.

**Nemenyi Test**

This test performs pairwise comparisons between all techniques to identify where the differences lie. This test is based on the concept of Critical Difference (CD), which represents the minimum difference in average ranks required for two techniques to be considered significantly different.

The mathematical formula to compute CD is represented as equation 2.23:

$$CD = q_\alpha \sqrt{\frac{k(k+1)}{6n}} \tag{2.23}$$

Where, $q_\alpha$ is critical value from studentized range statistic. The computed CD value is then compared with the absolute difference between the average ranks of two techniques. If this difference is greater than or equal to the CD value, the performances of the two techniques are considered to differ significantly at the chosen significance

level ($\alpha$) Conversely, if the difference is smaller than the CD value, the techniques are regarded as statistically equivalent in terms of performance.

## 2.13 Model Deployment

Model deployment is a critical stage in the ML pipeline, as it transitions predictive models from controlled experimental environments into practical, real-world applications. The primary objective of deployment is to evaluate not only the predictive capability of the models but also their feasibility in operational settings, where factors such as resource consumption, scalability, and security play significant roles.

In this thesis, the developed models were containerized using Docker to enable platform-independent deployment and reproducibility. Containerization ensures that the models, along with all necessary dependencies, can be executed consistently across diverse computing environments. This approach also facilitates integration with orchestration tools such as Kubernetes, which can manage workload distribution and dynamic scaling in CC infrastructures. During deployment, the following aspects were analyzed:

- Resource Consumption
- Security Vulnerability Analysis

## 2.14 Security Vulnerability Analysis

Modern software systems, particularly those deployed in containerized environments, are often susceptible to vulnerabilities that can compromise reliability, privacy, and trustworthiness. Predictive models developed in this thesis were therefore subjected to security vulnerability analysis to assess risks associated with deployment. The containerized environments were scanned using widely adopted tools, including *Anchore*, *Aqua Trivy*, *Docker Scan*, *Snyk*, *Docker Scout*, and *JFrog Xray*. A set of criteria was followed in the selection of container vulnerability scanning tools. We selected commonly used, open-source tools that have already been employed by researchers and

practitioners. The trial versions of paid products have limited functionalities, due to which free tools were taken into consideration. These tools examine container images for outdated packages, misconfigurations, and known vulnerabilities, categorizing them into low, medium, and high-severity risks. The focus was on identifying vulnerabilities such as denial-of-service (DoS) attacks, buffer overflows, privilege escalation, and insecure dependencies, which could undermine the integrity and reliability of the deployed systems. We installed tool extensions used in our study on Docker Desktop and, after that, the generated Docker image was analyzed to detect any security vulnerabilities.

**Anchore**

An open-source tool that inspects container images for security vulnerabilities, policy violations, and compliance issues. It provides detailed reports on packages, libraries, and dependencies.

**Aqua Trivy**

A comprehensive vulnerability scanner for containers and file systems. It detects Common Vulnerabilities and Exposures (CVEs) in operating systems, programming language libraries, and Infrastructure-as-Code (IaC) files.

**JFrog Xray**

A DevSecOps tool integrated with JFrog Artifactory. It performs deep recursive scanning of software components, including transitive dependencies, and provides contextual analysis for vulnerability impact.

**Snyk**

A developer-focused security tool that scans container images, application dependencies, and configuration files. It emphasizes actionable remediation by suggesting direct fixes and patches.

**Docker Scout**

A relatively new tool integrated into the Docker ecosystem. It provides visibility into container image dependencies and offers actionable insights on reducing vulnerabili-

ties at build and runtime.

**Docker Scan**

A built-in Docker CLI plugin that leverages Snyk's database to identify known CVEs in container images quickly, making it convenient for lightweight checks.

Since each tool has unique strengths and limitations, a combination of scanners was employed in this thesis to maximize vulnerability coverage and ensure more reliable deployment security analysis.

## 2.15 Resource Consumption Analysis

In addition to predictive accuracy and security, the practical deployment of ML models must be evaluated in terms of resource consumption. This ensures that the proposed solutions are not only effective but also computationally efficient and scalable in real-world environments such as cloud data centers. Resource consumption analysis focuses on the measurement of CPU usage, memory usage, disk utilization, and network overhead during the execution of containerized predictive models.

- **CPU Usage:** Percentage of CPU cycles consumed by the model, indicating computational intensity.
- **Memory Usage:** Amount of RAM used, critical for models with large parameters and batch sizes.
- **Disk I/O Usage:** Read/write operations generated, relevant for data-intensive tasks.
- **Network Usage:** Bandwidth consumption, important for federated learning and distributed deployment scenarios.

To effectively evaluate the deployment feasibility of the proposed models, resource consumption was monitored using a combination of cAdvisor, Prometheus, and Grafana,

which together form a widely adopted container monitoring stack.

**cAdvisor (Container Advisor)**

An open-source tool that provides real-time monitoring of resource usage (CPU, memory, disk, and network) at the container level. It collects performance metrics directly from Docker containers, enabling fine-grained analysis of how each model consumes resources during execution.

**Prometheus**

A time-series database and monitoring system used to scrape and store metrics exported by cAdvisor. It supports powerful querying through PromQL, allowing detailed trend analysis of system utilization under different workloads.

**Grafana**

A visualization and dashboarding tool integrated with Prometheus. Grafana provides intuitive and interactive dashboards, enabling visual interpretation of CPU, memory, and network consumption trends across experiments.

While cAdvisor collected raw container-level metrics, Prometheus stored them in a structured format, and Grafana facilitated visualization and comparative analysis across experiments. The combination ensured both quantitative accuracy and visual interpretability in assessing the efficiency of deployed models.

# Chapter 3

# SYSTEMATIC LITERATURE REVIEW

## 3.1 Introduction

The area of CC is gaining tremendous attention nowadays. It manages the resources dynamically and automatically. Without virtualization, there is no existence of CC as it reduces the hardware maintenance overhead [94]. We can achieve virtualization with the help of virtual machines and containers. Containers are replacing virtual machines as containers provide shared host operating system, fast deployment, quick launch time, portability, and scalability. A container is a closed environment for the software that bundles all binaries, library files, and supported files needed by the application in one package to provide a platform-independent run time environment that improves portability and productivity [8]. Containers are lightweight virtualization technologies. Containerization is the process of executing multiple applications on the same operating system or machine. In hypervisor-based virtualization, each application requires a separate operating system but in containerization, applications share an operating system, so the container-based virtualization technique is light weighted and deployment can be smaller in size. The main benefits of container-based virtualization are it virtu-

alizes operating system, takes a few seconds to run, is smaller in size, is lightweight, and requires less memory. The main drawback of containerization is containers are less secure than virtual machines. As containers use host operating system so if we restart the container then there is no need to reboot or restart the host operating system.

### 3.1.1 Containerization Technologies

In this section, we describe some container technologies such as docker, LXC, OpenVZ, singularity, and uDocker that represents container-based virtualization techniques. Now a days, these technologies are used by scientific computing communities. Container technology such as Docker focuses on specific industry applicability and singularity focuses on container portability across high performance computing environments. These technologies execute its own set of methods to achieve network isolation. Table 3.1 shows the comparative study of various containerization technologies such as docker, lxc, openvz, singularity, and udocker.

Table 3.1: Containerization Technologies

| **Docker** | **LXC (Linux Container)** | **Open VZ (Open Virtuozzo)** | **Singularity** | **uDocker** |
|---|---|---|---|---|
| Introduced by a company named as Docker, Inc. | Introduced by Daniel Lezcano and Serge Hallyn. | Introduced a company named SWsoft which is currently named as Parallels. | Introduced by Gregory M. Kurtzer. | Introduced by a team at CERN (European Organization for Nuclear Research) team. |
| It is used for application-level containerization. | It is used for system level containerization. | Same as LXC. It is used for system level containerization. | It is used for application-level containerization. | It is used for application-level containerization. |

| | | | | |
|---|---|---|---|---|
| Supported on all platforms (windows, linux, macOS) | Supported on linux platform only. | Supported on linux platform only. | Supported on linux platform only. | Supported on linux platforms only. |
| Enhanced version of LXC. Docker containers share host OS but runs in isolated user space. They are used in DevOps and application deployment scenarios. | LXC directly operates on Linux kernel and is used for running multiple containers on a single host. | It has its own kernel (Virtuozzo kernel) that it uses for containerization. Same as LXC, OpenVZ also runs multiple containers on a single host. | Singularity provides the facility to run the containers without requiring root access and used on high performance computing clusters and supercomputers. | uDocker is used where users need to execute Docker containers without requiring root access or other administrative rights. |
| Faster deployment speed | Deployment speed is slower as compared to docker and singularity | Deployment speed is slower as compared to docker and singularity | Deployment speed is slower as it takes longer startup time than docker | Deployment speed is slower than others. |

### 3.1.2 Container Orchestration Tools

In this section, we describe container orchestration tools. These tools are used to manage application deployment, load balancing, allocation of resources, scheduling, maintaining secure communication between containers, and scaling [27],[6]. Container-based virtualization uses many orchestration tools such as Docker swarm, Kubernetes, Mesos, Nextflow, Cloudify, Openshift, and Rancher. A workflow management system called Nextflow was created for data-driven computational pipelines. Nextflow is intended to provide a flexible, reproducible, and portable workflow management so-

lution running on multiple platforms such as cloud platforms and high-performance computing. Table 3.2 describes the comparative study of docker swarm, Kubernetes, and mesos.

Table 3.2: Comparison of Docker Swarm, Kubernetes, and Mesos

| **Docker Swarm** | **Kubernetes** | **Mesos** |
|---|---|---|
| Docker swarm was introduced by Docker. | Kubernetes was introduced by Google. | Mesos was introduced at University of California, Berkeley. |
| Easy to integrate and setup, flexible API, but limited customization. | Highly versatile, large open-source dev community, but more expensive. | Best for large—systems and designed for maximum redundancy. |
| Great value and easily scalable for small to medium systems. | Best for medium-scale, highly redundant systems, but requires a larger IT staff. | The most stable platform for large scale data solutions, but overly complex for small-scale systems under 10-20 nodes. |
| Running rate is limited to 1000 nodes. | Running rate is limited to 1000 nodes. | Can run upto 50000 nodes. |
| For simpler orchestration needs and smaller containerized applications, Docker Swarm is suitable. It is often selected when you want to incorporate clustered container management into your existing Docker deployment. | For complex microservices architectures and applications that require scalability, robustness, and fine-grained management of containerized workloads, Kubernetes is often used. | Mesos is appropriate for companies with a variety of workloads, including big data, containers, and other applications. It offers effective resource utilization for a variety of applications and resource isolation. |

Container technology has been used in many fields such as Scientific computing, big data processing, high-performance computing, high throughput computing, CC, fog computing, edge computing, the internet of things, DevOps, and grid computing. When there are only a few containers, container maintenance can be as simple as learning the build and run syntax for docker, signing up for a docker hub account, and launching your preferred text editor. However, if there are a large number of containers, it becomes necessary to maintain the software within the containers so that the

IT team can easily manage them, the container maintenance strategy is broadly classified into four components: container image supply chain, container infrastructure and orchestration, container runtime security, and container observability. The container image supply chain defines how the containers are built, which files to include in the container, how to assign unique tags to the images, and tools that check for logical errors and smells in the files. The second step describes where and how the containers run together. Orchestration tools such as Docker Swarm, Kubernetes, and Mesos are used to schedule the containers on working nodes. The third step, container runtime security, deals with approaches to detect vulnerabilities in running containers, and how to handle vulnerabilities that are detected. The last step, container observability, is related to runtime metrics such as CPU usage, memory usage, network bandwidth usage, and disk usage. The existing literature surveys [8], [22], [23] focused on container scheduling techniques, various methods used to analyze the performance of containers, benchmark tools used, and container performance monitoring tools. This paper aims at providing a summative evaluation of existing literature studies on containerization technology to provide a better understanding of container security attacks, vulnerabilities, approaches to mitigate security vulnerabilities and attacks, various container scheduling techniques used in the literature, approaches to identify which image to be included in the container, image tagging approaches, benchmark tools used to evaluate performance, and various container performance monitoring tools. The rest of the paper is assembled as follows: Method for conducting a literature review (Research methodology) is described in Section 2, which consists of defining research questions, inclusion and exclusion criteria, and quality assessment criteria. Section 3 describes the answer to the framed research questions. Section 4 describes the conclusion and future guidelines.

## 3.2 Research Methodology

This paper follows the guidelines suggested by [138],[186] which decomposes a systematic literature review into three phases: planning the review, conducting the review,

and reporting the review results. We identify the need for systematic review and develop protocols for review in the first phase. In the second phase, we frame research questions that are needed to be addressed in the systematic literature review, quality assessment criteria to analyze the studies, inclusion and exclusion criteria, and study selection criteria that includes identification of relevant keywords and identifying the primary sources or databases to be search in order to identify the relevant studies. In the last phase, we report the review results. In this section, we define the research questions underlying our survey, literature search strategy, quality assessment criteria, and inclusion and exclusion criteria.

### 3.2.1 Research Questions

Table 3.3 shows the research questions (RQs) formed based on the objectives described in the introduction section. From the selected primary studies, we analyzed how many studies are related to container image detection, container scheduling techniques, container security measures, and container performance evaluation (RQ1). We analyzed studies related to container image detection to answer RQ 2, RQ 2.1, RQ 2.2, and RQ 2.3. RQ 3, RQ 3.1, RQ 3.2 and RQ 3.3 assess which techniques and which performance objectives have been used in scheduling the containers. In RQ 4, we analyzed container security vulnerabilities, container security attacks, and various approaches used in mitigating the vulnerabilities and attacks. The last question RQ 5 assess the studies related to performance evaluation of containers, benchmarks tools, and container performance monitoring tools.

Table 3.3: Research Questions

| | |
|---|---|
| RQ 1 | How many studies exist in the literature which deal with mining the container images, scheduling the containers, container security, and evaluating the performance of the containers? |
| RQ 2 | Which container images, tagging approaches, smell detection approaches, and image mining approaches have been used in containerization? |
| RQ 2.1 | What are the approaches used in container image tagging? |

| | |
|---|---|
| RQ 2.2 | What are the various types of smell detected and the approaches used to detect the smell in container images? |
| RQ 2.3 | What are the image mining approaches used in literature? |
| RQ 3 | Which dataset, techniques, and performance objectives have been used in container scheduling? |
| RQ 3.1 | Which type of dataset has been used in container scheduling? |
| RQ 3.2 | What are the various performance objectives used in container scheduling? |
| RQ 3.3 | Which technique has been used in container scheduling? |
| RQ 4 | What are the various security vulnerabilities, vulnerability detection methods, attacks, and solutions to mitigate the vulnerability issues in containers? |
| RQ 4.1 | What are the various security vulnerabilities in containers and which methods have been used to detect the vulnerabilities? |
| RQ 4.2 | Which methods have been used to resolve the security vulnerability issues? |
| RQ 4.3 | What are the various types of attacks based on attack layers (container, application, docker engine, and host) and approaches used to detect security attacks in containerized applications? |
| RQ 5 | Which techniques, benchmarks, and monitoring tools have been used to evaluate the container performance? |
| RQ 5.1 | What are the various performance evaluation techniques used to evaluate the performance of the containers? |
| RQ 5.2 | Which benchmarks have been used in the performance evaluation of containers? |
| RQ 5.2.1 | What are the various CPU-intensive benchmarks? |
| RQ 5.2.2 | What are the various memory-intensive benchmarks? |
| RQ 5.2.3 | What are the various disk I/O intensive benchmarks? |
| RQ 5.2.4 | What are the various network I/O intensive benchmarks? |
| RQ 5.3 | Which tools have been used to monitor the container performance? |

### 3.2.2 Search Strategy

We collect the primary studies from various scientific literature databases by using the search string ”(Container OR Docker) AND (Scheduling OR Resource allocation OR Heuristic OR Metaheuristic OR Mathematical modeling OR Machine learning) AND Performance evaluation AND Monitoring tool AND Benchmark tool AND (Security OR Vulnerability OR Attacks) AND Image detection AND (Smells OR Antipatterns) AND Tagging”. Here, **AND** combines the main search terms, and **OR** combines alter-

native terms and synonyms. After selecting the search terms, we identified the scientific literature databases used in this study [186]. The following databases were used to search for relevant studies:

1. IEEE Xplore
2. Springer
3. ACM Digital Library
4. Google Scholar
5. Elsevier
6. Wiley
7. Taylor & Francis

After choosing which databases to search, a preliminary search was conducted to find the primary studies. After performing this preliminary search, the pertinent studies were identified by obtaining the full-text papers, following the inclusion and exclusion criteria outlined in the next section. We also included those studies that were found important from the reference sections of the selected primary studies. The final studies were chosen after applying the defined inclusion and exclusion criteria. We identified 140 primary studies initially. After reviewing the reference list of the pertinent studies, further 40 studies were included. Hence, total number of 180 studies were identified for further analysis.

### 3.2.3 Inclusion and Exclusion criteria

The following inclusion and exclusion criteria were followed for selecting studies in this systematic literature review:

**Inclusion Criteria:**

- The study must be related to computer science and engineering and propose techniques, methods, and tools for container-related problems.
- The study must be written in English.
- The study must pass quality assessment criteria.
- If a conference paper has a significant journal extension, then both the conference and journal publications will be included.

**Exclusion Criteria:**

- Studies that are not related to the field of computer science and engineering.
- Thesis, poster presentation papers, demo papers, and short papers.

By applying the above inclusion and exclusion criteria, a total of 160 studies were selected. The quality assessment criteria given in Section **??** were used to identify the final primary studies.

### 3.2.4 Quality Assessment Questions

We formed quality assessment questions for weighing the studies as. These questions are ranked 1 or 0.5 or 0. The final score is calculated by adding the score assigned to each question. A study could score a maximum of 9 and a minimum of 0. Studies having a final score of 5 or more than 5 will be included in primary studies. After applying quality assessment criteria, we identified 145 primary studies. Table 3.4 shows quality assessment questionnaire.

## 3.3 Review Results

In this section, we will answer all the defined RQs based on selected primary studies. The answers to the defined research questions are given in each sub sections. The primary studies are categorized based on publication type and publication year as shown

Table 3.4: Quality Assessment Criteria

| QA | Question |
|---|---|
| QA 1 | Does the study state the main objectives and research questions in a clear way? |
| QA 2 | Does the study clearly give the details about selected projects? |
| QA 3 | Is the dataset clearly described (number of stars, contributors, services, and containers)? |
| QA 4 | Does the study give a detailed description of selected techniques? |
| QA 5 | Does the study clearly define the parameter settings of the techniques used? |
| QA 6 | Are the performance objectives clearly stated? |
| QA 7 | Does the study clearly report the results? |
| QA 8 | Are the result compared with the results of previous studies? |
| QA 9 | Does the study discuss about validity threats? |

in figure 3.1. It shows that majority of the studies were published between 2017 and 2022. Out of 145 primary studies, 48 studies were published in top most journals such as IEEE Access, Future Generation Computer Systems, Cluster Computing, Soft Computing, etc., 87 studies were published in top conferences, and 10 studies were published as book chapters. Figure 3.2 depicts the distribution of primary studies by their publishers over the year between 2015 and 2022. It shows that majority of the studies have been published at IEEE, Springer, and ACM.

### 3.3.1 RQ 1 Results

Table 3.5: Classification of Studies

| | |
|---|---|
| Image Detection | [40], [44], [52], [53], [96], [106], [165], [180], [241], [242], [276], [307], [308], [312], [319], [318], [334], [338]. |
| Container Scheduling Techniques | [7],[14], [15], [210], [22], [41], [48], [68], [73], [74], [86], [87], [89], [97], [103], [104], [108], [123], [127], [129], [128], [131], [157], [158], [161], [162], [166], [172], [171], [177], [179], [182], [183], [194], [198], [201], [202], [204], [203], [205], [219], [221], [224], [230], [237], [245], [246], [249], [255], [262], [268], [271], [269], [270], [272], [274], [289], [306], [311], [326], [332], [333], [339]. |
| Container Security | [18], [78], [93], [105], [115], [119], [125], [152], [155], [167], [168], [175], [178], [195], [207], [235], [256], [265], [267], [280], [284], [299], [300], [323], [322], [321], [328], [337]. |

| | |
|---|---|
| Performance evaluation of containers | [3], [4], [5], [24], [25], [28], [29], [36], [37], [38], [46], [116], [117], [120], [141], [154], [164], [176], [185], [193], [197], [196], [200], [212], [213], [214], [217], [229], [234], [236], [238], [248], [252], [293], [317], [320], [329]. |

Table 3.5 shows the primary studies related to each phase, container image detection, container scheduling, container security, and container performance analysis. Figure 3.3 describes pie chart of selected primary studies in each phase. This figure shows that out of 145 studies, 24% studies are related to performance evaluation of containers, 42% studies are related to container scheduling approaches, 22% studies are related to container security, and 12% studies are related to container image detection.

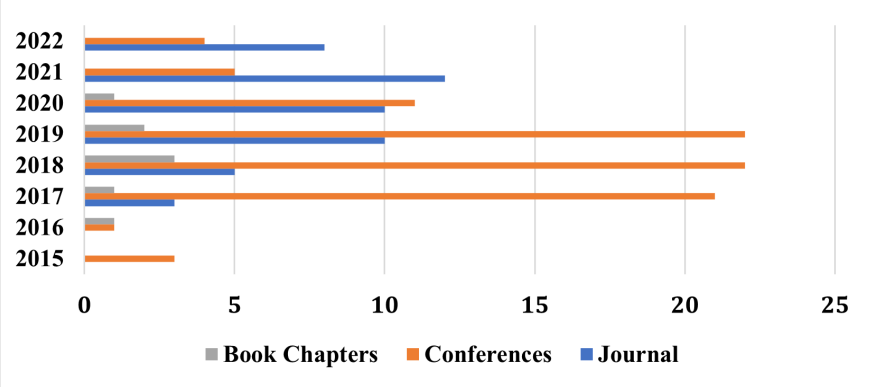



Figure 3.1: Categorization based on Publication Type and Publication Year

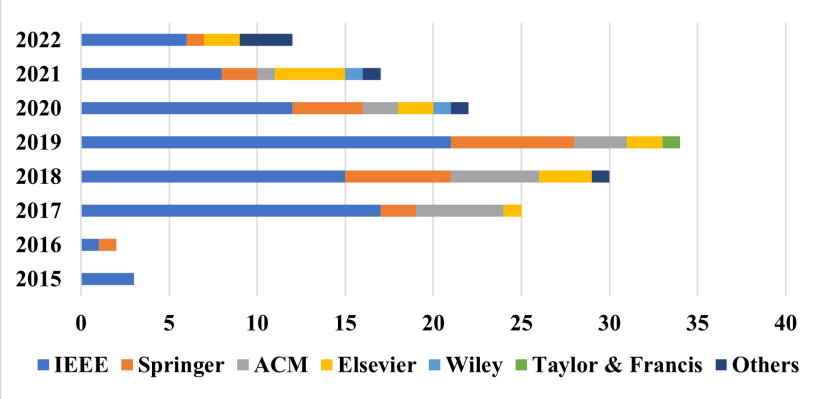



Figure 3.2: Distribution of Primary Studies by Publisher Over Years

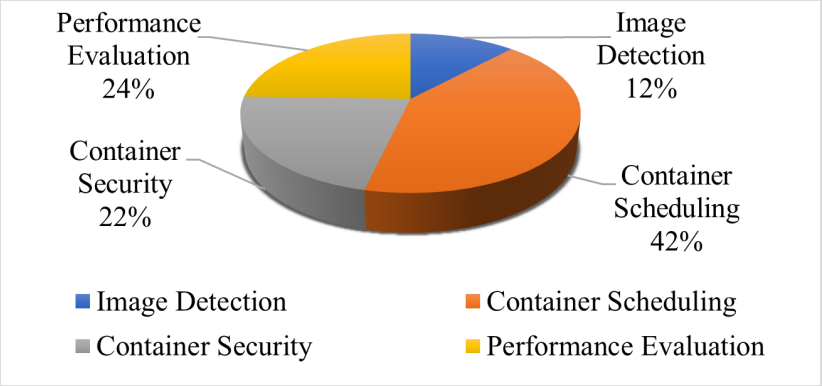



Figure 3.3: Primary Studies Selected in Each Phase

**Discussion**

- According to the analyzed studies, most papers were published between 2017 and 2022.
- Most of the studies were published in conference proceedings. Out of 145 studies, 89 studies were published as conference proceedings, 48 studies were published in journals, and 8 studies were published as book chapters.
- We have described 4 phases of software container maintenance: container image detection, container scheduling, container security, and container performance evaluation.
- Container scheduling is the most explored area in the analyzed papers. Out of 145 studies, 62 studies are related to container scheduling, 37 studies are related to container performance evaluation, 28 studies are related to container security, and 18 studies are related to container image detection.

### 3.3.2 RQ 2 Results

In this section, we explain image mining approaches, smell detection approaches, and tagging approaches in containerizing an application. Lin *et al.* [165] conducted an

empirical study in which they explored base images, popular programming languages, some tagging practices, and various code smells, and created a dataset by collecting data from GitHub, Docker Hub, and Bitbucket repositories. The authors used release tagging and SHA hash pinning approaches, and the results show that the release tagging approach is more commonly used, and the number of code smells in Dockerfiles is decreasing day by day, which indicates an improvement in the quality of Docker images. Cito *et al.* [52] explored smells in Dockerfiles and popular base images. This study was the first of its kind on Dockerfiles and highlighted various features of the Docker ecosystem, considering Dockerfiles stored in the GitHub repository. Ibrahim *et al.* [106] explored the differences between official and community Docker images for five software systems. They considered 936 images for analysis, which consisted of 929 community images and 7 official images. The results of this study demonstrate that 49%, 26%, and 8% of the community Debian, Alpine, and Ubuntu images use fewer resources than their respective official images. Furthermore, compared to official images, 7% of the community images have fewer security flaws.

**Container Image Tagging Approaches (RQ2.1)**

There can be a large number of Docker repositories that contain Docker images and Dockerfiles. So, there is a need to assign some tags to Docker repositories so that necessary files can be easily retrieved. Manually assigning tags is a time-consuming and exhausting process. Yin *et al.* [318] proposed D-Tagger, which is a tag recommendation approach based on the Labeled Latent Dirichlet Allocation (L-LDA) algorithm. It recommends tags to Docker repositories based on their Dockerfiles. The results show that D-Tagger achieves 0.675 accuracy in terms of Recall@5 and 0.712 accuracy in terms of Recall@10. Yin *et al.* [319] also proposed STAR (Specialized Tagging Approach for Docker Repository), which is used for recommending tags. The approach combines L-LDA and similarity-based ranking methods for tag recommendation. The outcome of this study shows that the proposed approach achieves 0.675 and 0.712 accuracy in terms of Recall@10 and Recall@5, respectively. Chen *et al.* [44] proposed a semi-supervised tagging method called SemiTagRec, a tag recommendation approach for Docker repositories. The proposed approach consists of four components: *Predic-*

*tor*, *Extender*, *Evaluator*, and *Integrator*. The results of this study depict that SemiTagRec outperforms STAR [319] and EnTagRec in terms of Recall@10 and Recall@5, with accuracies of 0.781 and 0.688, respectively.

**Types of Smells and Approaches Used to Detect Them (RQ2.2)**

To import data and applications during the image building process in containers, temporary files are used. If these temporary files exist in container images, it may lead to an increase in image size and thus effect QoS. These are called temporary file smells. Xu *et al.* [312] proposed two approaches with static and dynamic analysis to detect temporary file smells. Approach based on dynamic analysis consists of two steps: code injection and log analysis. Approach based on static analysis consists of three steps: building the abstract syntax tree, semantic analysis, and file matching. The result shows that the dynamic approach detects 100% temporary file smell and the static approach is a lightweight and easy-to-use approach. Lu *et al.* [180] proposed a state-dependent static analysis method to detect temporary files. They also proposed 3 approaches to remove detected temporary file smells: instruction merge method, directly copy method, and external storage method. Experimental result shows that the proposed approach effectively and efficiently detect and remove temporary file smells. Wu *et al.* [308] conducted an empirical study that helps developers to gain some information related to the presence of Docker file smells including its distribution, correlation with project characteristics, coverage, and co-occurrence. They used linear regression to identify the relationship between Docker file smells and project characteristics.

**Image Mining Approaches (RQ 2.3)**

Application containers such as docker are gaining popularity these days for software deployment. They can pack irrelevant resources which may lead to an increase in space usage and less security. Rastogi *et al.* [241] proposed CIMPLIFIER as a solution to detect necessary files in a container which is named as a container image. CIMPLIFIER partitions the container into simple ones and excludes those resources which are not needed to execute the application. CIMPLIFIER follows three steps: resource

identification, container partitioning, and gluing. The result of this study depicts that proposed method reduces container image size up to 95% and processes even large containers in less than thirty seconds. As in dynamic analysis, there can be a possibility of inaccuracy in resource set identification because of incomplete code coverage during container execution, and in the case of static analysis approach, there can be an incompatibility in languages. So, Rastogi *et al.* [242] combine both static and dynamic analysis approaches to build an intermediate representation for container debloating. Civolani *et al.* [53] proposed FogDocker which reduces the container start-up time on both fog computing server and CC server by finding the necessary files to start the container and deploying these files firstly. The goal of this study is to significantly reduce the time spent needed to deploy the container until the container is ready to perform relevant functionality. Lack of knowledge and experience may lead to the development of Docker files with more quality issues. Hassan *et al.* [96] proposed RUDSEA which gives information about update locations in Docker files and also recommends changes in docker files based on changes in software environment factors. The outcome of this study depicts that the proposed approach gives correct updates for 44.1% of updates and correct update location for 78.5% of updates.

**Discussion**

- We discussed about various tagging approaches, various types of code smells and approaches used to detect code smells, and the frameworks introduced to detect the necessary files in a container.
- As there can be a large number of docker repositories that contains docker images, the authors introduced various methods such as D-Tagger, STAR, EnTagRec, and SemiTagRec to assign unique tags to them so that they can be easily retrieved.
- Image detection is the process to identify the necessary files to create the container image so that it consumes less space.

### 3.3.3 RQ 3 Results

There is a need to introduce an efficient and effective approach to allocate nodes to execute a set of containers so that it takes less time and reduces performance overhead such as energy consumption, network bandwidth, cost, etc. In this, we will discuss about dataset, performance objectives, and various container scheduling techniques (heuristic, metaheuristic, mathematical modeling, and machine learning) used in the literature.

**Dataset (RQ 3.1)**

Some authors have used industrial datasets such as online submitted containers requests, and some have used open-source datasets such as docker files, nginx, tomcat, debian, ubuntu, alpine, Kubernetes, etc. Table 3.6 categorizes the primary studies according to the dataset used.

Table 3.6: Dataset Used

| | |
|---|---|
| **Industrial dataset** | [183], [202], [204], [203], [205], [212]. |
| **Open-source dataset** | [7], [22], [86], [97], [103], [108], [123], [129], [158], [161], [162], [166], [172], [194], [198], [201], [237], [255], [262], [271], [269], [270], [272], [274], [289], [306], [339]. |

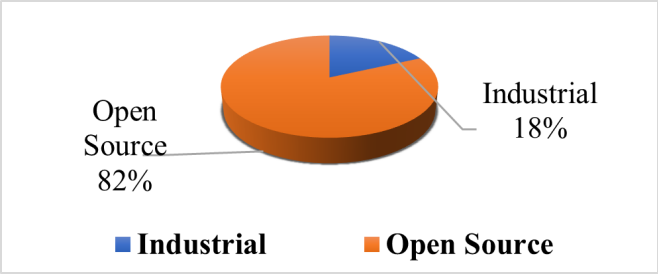



Figure 3.4: Dataset Used

Figure 3.4 describes that in 82% studies, authors have used open-source datasets and in 18% studies, authors have used industrial dataset. Most of the studies in the literature have used open-source dataset. Auver grid trace, google trace data, nginx server images are mostly used open-source dataset in container scheduling techniques.

**Performance Objectives (RQ 3.2)**

Various types of performance objectives such as load balancing, resource utilization, makespan, energy consumption, scalability, cost, etc. have been used in the literature to analyze the performance of container scheduling techniques. An optimization objective is used to access the quality of the solution provided by an algorithm. Optimization objectives can be single objective or multi-objective depending on the optimization required for the algorithm [8]. The following performance objectives shown in table 3.7 are frequently used to measure the performance of the scheduling technique.

Table 3.7: Performance Objectives Used in Container Scheduling

| **Performance Objective** | **Description** | **Studies** | **Percentage** |
|---|---|---|---|
| Resource Utilization | It describes how efficiently the available resources (memory and network bandwidth) are utilized on working nodes. Higher the resource utilization of nodes, the node becomes more energy efficient and cost effective. | [7], [14], [15], [210], [22], [73], [74], [87], [103], [104], [108], [123], [127], [129], [131], [158], [161], [166], [172], [171], [183], [194], [198], [195], [219], [237], [245], [249], [255], [262], [271], [274], [289], [311], [333], [339] | 58.06% |
| Scalability | It deals with the capacity to continuously provide the necessary services despite an increase in load on the system from a single application or from several. | [68], [103], [108], [194], [268] | 8.06% |
| Availability | It indicates the amount of time that a user can access a particular application or service anytime they want to do so. | [14], [15], [74], [87], [108], [202], [270], [306] | 14.52% |

| | | | |
|---|---|---|---|
| Load Balancing | It describes the equal allocation of workload on working nodes so that no working nodes becomes overloaded while other nodes are idle. | [15], [74], [89], [103], [123], [127], [129], [128], [157], [161], [162], [166], [171], [179], [182], [183], [194], [201], [204], [203], [205], [237], [262], [274], [311] | 40.32% |
| Cost | The total cost of an application comprises three aspects: computation, communication, and storage. Computation cost is the time taken to execute an application on available nodes and communication cost is the amount given to network providers for telecommunication services that are required to execute an application. | [73], [86], [177], [183], [221], [255], [289], [326], [339] | 14.52% |
| Energy Consumption | It describes how much energy is consumed when containers are deployed on working nodes. We try to reduce the overall energy consumption rate to lower the computation cost. | [7], [210], [41], [97], [108], [129], [128], [131], [157], [177], [198], [201], [203], [205], [224], [245], [271], [269], [270], [272], [326], [332] | 35.48% |
| Makespan | It describes the amount of time needed to execute the application or service from beginning to end. An effective scheduler tries to minimize the overall time. | [7], [210], [22], [74], [89], [104], [131], [157], [162], [171], [179], [183], [204], [203], [221], [246], [249], [255], [306]. | 30.64% |

| | | | |
|---|---|---|---|
| Network Transmission | It refers to the number of bits transferred per second. This objective helps in identifying communication delay between containers and also helps in managing network congestion. | [87], [166], [194], [245], [246], [326]. | 9.68% |

**Techniques (RQ 3.3)**

Containers are replacing traditional VMs as containers share OS, provide fast deployment, and are light weight so they are gaining tremendous popularity nowadays. The problem of efficiently and effectively allocating a node to execute a set of containers is a challenging task. Kubernetes and Docker Swarm orchestration platforms consist of three default container scheduling approaches to deploy the containers on computing nodes. Random strategy schedules the containers on any node randomly, binpack strategy schedules the containers on the node with the most containers, and spread strategy schedules the containers on nodes with the fewest containers. Various scheduling methodologies have been introduced in the literature. Omogbai Oleghe [230] presented a review of the container scheduling strategy used in edge computing. This study mainly focused on techniques and frameworks that have been used in developing container placement algorithms for edge computing problems. The study demonstrates that optimization modelling frameworks and heuristic-based algorithms are pervasive. Tan *et al.* [268] compared three heuristic approaches (sub, sum, and random) to show their inability to handle resource requirements well. After this, the author proposed a learning approach named as GPHH to design various rules for container allocation. The result depicts that GPHH outperforms sub, sum, and random approaches in terms of energy consumption. Container scheduling techniques are broadly classified into four categories: Heuristic, Meta- heuristic, Mathematical modelling, and Machine learning.

Figure 3.5 shows pie chart of container scheduling techniques. This figure describes that 33% studies are related to heuristic approaches, 32% studies are related to meta-heuristic approaches, 18% studies are related to mathematical modeling approaches,

and 17% studies are related to M: approaches.

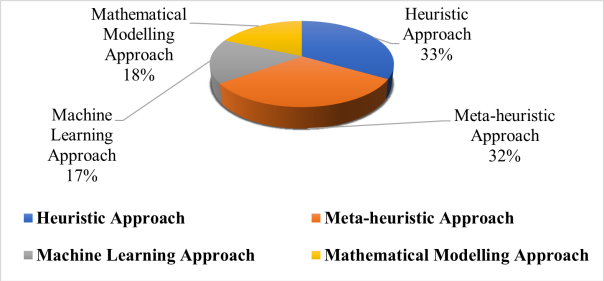



Figure 3.5: Scheduling Techniques

Several heuristic algorithms have been introduced in the literature to solve container scheduling problems. Some of them uses bin packing technique. Most of the proposed approaches use a combination of existing strategies in docker swarm and Kubernetes. Hu *et al.* [104] introduced ECSched (Enhanced Container Scheduler), an efficient scheduling approach to handle concurrent container requests on heterogeneous clusters. It follows a graph-based approach. This approach first fetches a set of container requests and then applies MCFP (Minimum cost flow problem) algorithm to efficiently place the containers. Experimental result shows that ECSched outperforms docker swarm and kubernetes default scheduling strategy in terms of average container completion time and resource utilization. We have presented a summary of findings related to heuristic approaches in solving container scheduling problems in table 3.8.

Table 3.8: Summary of Studies Related to Heuristic Approaches for Container Scheduling

| **Studies** | **Performance Objective** | **Findings** | **Techniques Used** | **Container Used** |
|---|---|---|---|---|
| [202] | Availability | Proposed a new multi-objective scheduling technique which is based on the combination of KUNG and PROMETHEE and used to schedule the containers to a node such that it should consider two objectives: no. of available CPU and memory size. | Multi-objective container scheduling algorithm based on multi-criteria decision analysis. | Docker |

| | | | | |
|---|---|---|---|---|
| [202] | Availability | Proposed a new multi-objective scheduling technique which is based on the combination of KUNG and PROMETHEE and used to schedule the containers to a node such that it should consider two objectives: no. of available CPU and memory size. | Multi-objective container scheduling algorithm based on multi-criteria decision analysis. | Docker |
| [127] | Resource Utilization, Load Balancing | Introduced a multi-dimensional container resource allocation algorithm, CRADLE+ which is designed using a local ratio algorithm. This algorithm allocates nodes or machines to execute the containers to get maximum profit. The result shows that this algorithm achieves 10-50% better results than binpack and spread. | CRADLE* using local ratio algorithm. | Kubernetes |
| [204] | Load Balancing, Makespan | Introduced a scheduling strategy by combining spread with binpack using TOPSIS algorithm to select the node that executes container which minimizes the following three criteria: No. of containers in each node, no. of CPUs, and memory size. | Spread=Binpack, TOPSIS (multi-criteria decision algorithm). | Docker |

| | | | | |
|---|---|---|---|---|
| [162] | Load Balance, Mean Squared Error, Makespan | In some cases, swarm not give good performance of load balancing. So, here a new scheduling strategy has been introduced that can meet various scheduling requirements. Result shows that MCSS strategy performs better in load balancing. The MSE results of MCSS are better than the other strategies. And has an average performance in terms of response time. | Multi algorithm collaboration scheduling strategy (MCSS) | Docker |
| [73] | Resource Utilization, Cost | To maximize the overall value gained from a workload, a market-value-inspired algorithm for container allocation in Docker Swarm has been proposed. In their scheme, container's execution value varies with time determined by the value curve of the job. | Market-based heuristic approach | Docker |
| [103] | Load Balancing, Resource Utilization, Scalability | Proposed a container scheduling algorithm with multiple objectives cost functions based on vector bin packing. The objectives include resource usage, load balancing, and network dependency. This algorithm improves overall performance by colocating containers with dependencies on the same host. | Vector bin packing algorithm | Kubernetes |

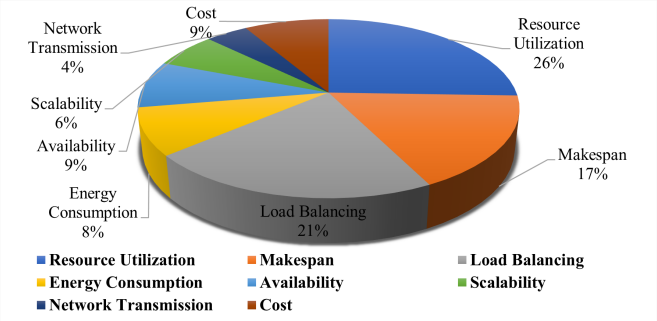



Figure 3.6: Performance Objectives Used in Heuristic Approaches

Figure 3.6 shows pie chart of performance objectives used in heuristic approaches for scheduling the containers on working nodes. This figure describes that among all the performance objectives, resource utilization is the most considered performance objective and network transmission is the least considered performance objective in heuristic approaches. Out of 20 studies, 26% studies used resource utilization, 21% studies used load balancing, 17% studies used makespan, 9% studies used cost, 8% studies used energy consumption, 9% studies used availability, 6% studies used scalability, and only 4% studies used network transmission. Nowadays, researchers are focusing on metaheuristic algorithms to solve optimization problems in diverse fields. Basically, metaheuristic algorithms are suitable for solving multi-objective optimization problems. Genetic algorithms, particle swarm optimization algorithms, ant colony optimization algorithms, whale optimization algorithms are some examples of metaheuristic algorithms. Tan *et al.* [271] proposed new genetic operators, a genetic algorithm to solve two-level container allocation problem, and compared the effectiveness of the proposed GA with best fit descending algorithm and single chromosome GA. The result shows that the proposed algorithm outperforms best fit descending and single chromosome GA in terms of both resource allocation and energy consumption. We have presented a summary of findings related to metaheuristic approaches in solving container scheduling problems in table 3.9.

Table 3.9: Findings Related to Meta Heuristic Approaches for Container Scheduling

| **Studies** | **Performance Objective** | **Findings** | **Techniques Used** | **Container Used** |
|---|---|---|---|---|
| [172] | Resource Utilization | The result shows that K-PSO algorithm increases | K-PSO algorithm | Kubernetes, Docker |

| | | | | |
|---|---|---|---|---|
| | | resource utilization by 20% as compared to PSO algorithm, ESS strategy, Kube scheduler default policy, QoS policy. | implemented on Kubernetes platform | |
| [108] | Availability, Reliability, Scalability, Energy Consumption, Resource Utilization | The result shows that many objective genetic algorithm schedulers improve the result as compared to Ant Colony Optimization algorithm in terms of task distribution, power consumption, unique task assignment, availability, and resource utilization. | Many objective genetic algorithms based on NSGA III algorithm | Docker |
| [89] | Load Balancing, Makespan | A container scheduling strategy based on neighbourhood division in micro service (CSBND) is proposed. Compared to the MOPSO algorithm and widely used Spread strategy in | CSBND integrated with PSO micro services, it can increase performance by 20% to 25%. | CloudSim |
| [161] | Resource Utilization, Load Balancing | Developed particle swarm optimization algorithm to schedule the docker containers to improve resource usage and load balancing. The proposed scheduling approach outperforms spread and random strategies. | PSO (Particle Swarm Optimization) algorithm | Docker |
| [249] | Makespan, Cost, Resource Utilization, SLA Violations | Used Ant Colony Optimization Algorithm to solve multi objective resource scheduling problem in order to maximize resource utilization with minimizing execution cost, time, SLA violations and space complexity. | Ant Colony Optimization Algorithm | Container Cloud |

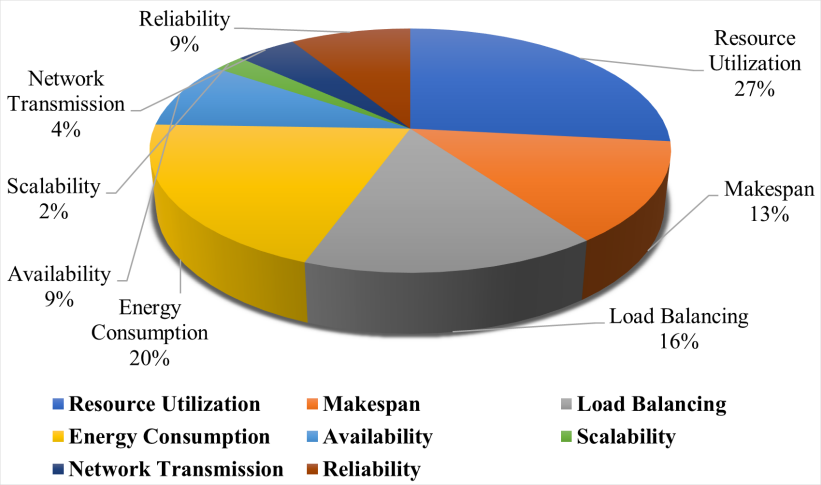



Figure 3.7: Performance Objectives Used in Meta Heuristic Approaches

Figure 3.7 shows pie chart of performance objectives used in meta-heuristic approaches for scheduling the containers on working nodes. This figure describes that among all the performance objectives, resource utilization is the most considered performance objective and scalability is the least considered performance objective in meta-heuristic approaches. Out of 19 studies, 27% studies used resource utilization, 10% studies used load balancing, 14% studies used makespan, 22% studies used energy consumption, 10% studies used availability, 5% studies used network transmission, 10% studies used reliability, and only 3% studies used scalability.

Integer Linear Programming (ILP) is a mathematical modelling technique which makes use of a set of linear constraints to optimize linear functions. The primary technique to solve the problems modeled using ILP formulation are branch and bound methods and CPLEX. The ILP formulation of container scheduling problem is NP complete in complexity. Guan *et al.* [86] developed an application oriented docker container (AODC) framework for resource allocation and formulate resource allocation problem as an optimization problem. The author further proposed a scheduling approach to address AODC optimization problem using linear programming solver (CPLEX). Experimental result shows that AODC framework and algorithm outperform worst fit, best fit, and optimal-VM (VM placement by CPLEX) approaches in terms of cost and acceptance ratio. A summary of findings related to mathematical modeling approaches in solving container scheduling problems is presented in table 3.10.

Table 3.10: Findings Related to Mathematical Modeling Approaches for Container Scheduling

| Studies | Performance Objective | Findings | Techniques Used | Container Used |
|---|---|---|---|---|
| [129] | Resource Utilization, Energy Consumption, Load Balancing | Introduced Kubernetes based Energy and Interference Driven Scheduler (KEIDS). The proposed scheduler gives an overall improvement of 14.42%, 47%, and 31.83%, respectively against the First come first scheduling technique in terms of energy consumption, Carbon footprint, and load balancing. | KEIDS platform which utilizes Integer linear programming based on multi objective optimization algorithm, Mosek solver for ILP | Kubernetes |
| [289] | Resource Utilization, Cost | Proposed a scalable and communication-efficient resource allocation algorithm, named EPTA, to reduce the application deployment cost-constrained by capacity and the service delay bound. To validate the effectiveness, the authors have compared the results of the proposed algorithm with existing approaches in docker swarm and the result depicts that the proposed algorithm is more flexible and less costly. | EPTA (Execution container Placement and Task sPherabili algorithm) which uses linear programming solver (e.g.: CPLEX) | Docker |
| [128] | Energy Consumption, Load Balancing | Designed multi objective optimization problem (MOOP) which has been expressed as ILP for scheduling containers and proposed an algorithm, EnLoB to solve MOOP to reduce | ILP (Integer Linear Programming), EnLoB (Energy and Load Balancing) | CloudSim |

| | | | | |
|---|---|---|---|---|
| | | energy consumption and balance the load among nodes. | | |
| [221] | Cost, Makespan | Proposed EVCD, a general formulation of the container deployment problem for Cloud environments that takes into account the heterogeneity of container requirements and computing resources. The experimental result indicates that EVCD can be used as a benchmark framework against first fit algorithm and round robin algorithm. | EVCD (Elastic provisioning of Virtual machines for Container Deployment) | Docker |

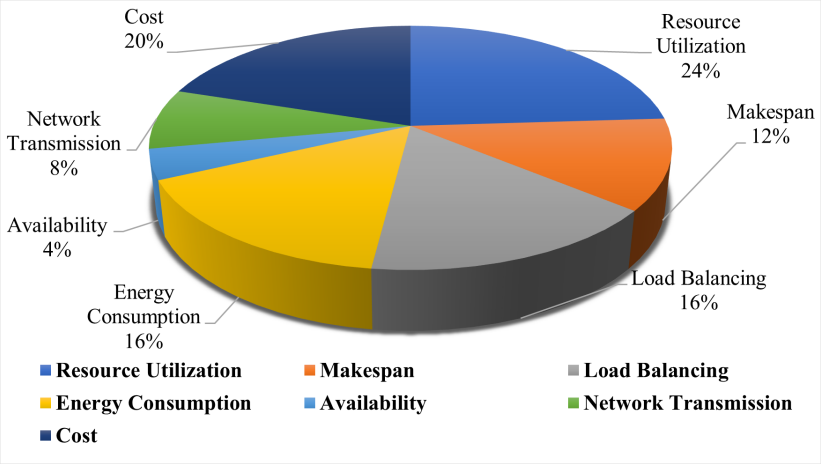



Figure 3.8: Performance Objectives Used in Mathematical Modeling Approaches

Figure 3.8 shows pie chart of performance objectives used in mathematical modeling approaches for scheduling the containers on working nodes. This figure describes that among all the performance objectives, resource utilization is the most considered performance objective and availability and cost is the least considered performance objective in mathematical modeling approaches. Out of 11 studies, 29% studies used resource utilization, 19% studies used load balancing, 14% studies used makespan, 19% studies used energy consumption, 5% studies used cost, 9% studies used network

transmission, and only 5% studies used availability.

Machine learning algorithms gather knowledge from data and act accordingly. A subfield of ML called DL uses a layered structure of ANN to learn and make intelligent decisions on its own. Nanda and Hacker [219] formulated a multi-resource bin packing problem and proposed Resource aware container consolidation (RACC) approach, Fit for Packing which uses deep reinforcement learning approach to place containers on physical machine. The result shows that our model achieves an average training accuracy of 82.01% and an average testing accuracy of 82.93% as compared to SJF and random strategy. A summary of findings related to ML approaches in solving container scheduling problems is given in table 3.11.

Table 3.11: Findings Related to ML Approaches for Container Scheduling

| **Studies** | **Performance Objective** | **Findings** | **Techniques Used** | **Container Used** |
|---|---|---|---|---|
| [203] | Energy Consumption, Makespan, Load Balancing | This paper addressed the problem of a company that receives containers from their user daily. The idea is to reduce power consumption while satisfying the need of all the users. For this, a new consolidation approach that is based on ML is proposed. | Consolidation approach based on ML technique (K-means) | Docker |
| [205] | Energy Consumption, Load Balancing | A new scheduling technique based on power efficiency is developed by improving the default swarm kit spread strategy. Power consumption gets reduced as compared to default swarm kit spread strategy. | Power consumption-based new container scheduling strategy which uses ML technique (K-means) | Docker |
| [182] | Load Balancing | Proposed a new container scheduling technique based on machine learning technique. It maintains load balancing and improves system performance. As | Random forest regression technique, Jmeter for load balancing and Prometheus | Kubernetes |

| | | | | |
|---|---|---|---|---|
| | | compared to the feedback technique, it gives a better result. | tool to monitor the performance | |
| [311] | Load Balancing, Resource Utilization | On the basis of container and service mesh technologies, a practical approach is introduced to create a flexible and on-demand system for deep learning applications. And the CPU utilization is 15.52% and the current cluster memory utilization counts for 76% of the overall memory. | Deep Learning, Grafana tool | Docker, Kubernetes |

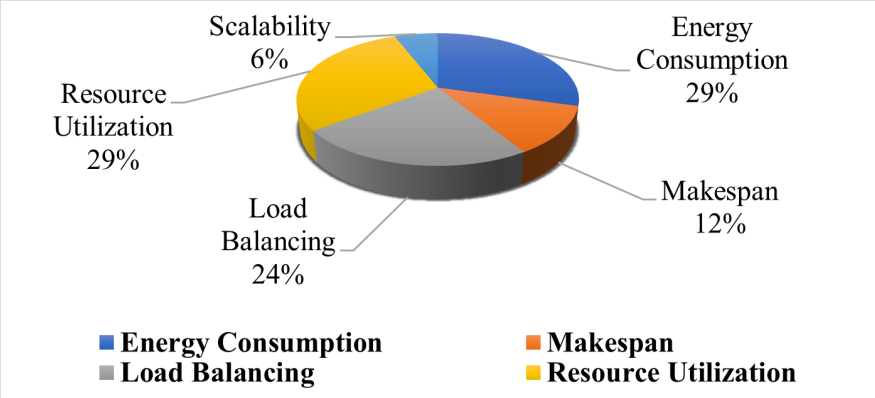



Figure 3.9: Performance Objectives Used in ML Approaches

Figure 3.9 shows pie chart of performance objectives used in machine learning approaches for scheduling the containers on working nodes. This figure describes that among all the performance objectives, resource utilization and energy consumption is the most considered performance objective and scalability is the least considered performance objective in machine learning approaches. Out of 10 studies, 29% studies used resource utilization, 24% studies used load balancing, 29% studies used energy consumption, 12% studies used makespan, and only 6% studies used scalability.

Discussion

- In the analyzed papers, open-source dataset such as Google Trace, Alibaba Cluster Trace is mostly used.
- Resource utilization, energy consumption, load balancing, makespan, etc. are performance objectives considered in container scheduling approaches. Resource utilization is the most used performance objective.
- Heuristic, meta heuristic, machine learning, and mathematical modeling approaches are used for scheduling containers on working nodes. Heuristic is the most used approach in the analyzed papers.
- Our analysis shows that out of 61 studies related to container scheduling approaches, 33% studies have used heuristic approaches, 32% studies are related to meta heuristic approaches, 18% studies have used mathematical modeling approaches, and 17% studies have used machine learning approaches.

### 3.3.4 RQ 4 Results

As containers follow kernel sharing property, there is a need to evaluate container security measures. In the literature, authors have identified various security issues and proposed various defense methods and approaches to detect and resolve security vulnerabilities in containers. Lin *et al.* [167] in their study collected a dataset of 223 exploits from Exploit-db database that may be helpful in attacking containers and categorized them into two parts using two-dimensional taxonomy. First dimension classification is based on the consequences of attacks and second dimension classification is based on the layers at which vulnerable program resides. They also analyzed privilege escalation attacks on containers and proposed a defense method to easily defend the detected privilege escalation attacks. Zerouali *et al.* [321] presented an empirical study on outdatedness and vulnerabilities due to third-party packages. They mainly focused on javascript, ruby, and python packages installed in 3000 docker hub community images. These packages may be outdated or missed some releases which lead to potential

vulnerabilities in images. The result of this study shows that the maintainers should invest more time and effort in updating third-party packages in order to reduce vulnerabilities in docker hub images. Zerouali *et al.* [323] analyzed the impact of npm packages in docker images. They analyzed outdatedness and vulnerability of npm packages in docker environment. The result of this study depicts that npm packages are outdated and thus may result in potential security vulnerabilities in docker images. Liu *et al.* [175] presented an empirical study to understand the security threats in docker environment and identified 3 major sources of security risk in docker containers: sensitive parameters in run commands, malicious docker images, and vulnerabilities in docker images. Authors also suggest solutions to uncover identified threats such as detecting malicious images, automatically resolving vulnerability issues in images, etc.

**Security Vulnerabilities and Approaches to Detect it (RQ 4.1)**

Martin *et al.*[195] performed vulnerability-oriented analysis to examine the security issues associated with the use of containers in different use cases. They provided a thorough survey of existing studies in the field of Docker containers and then created a detailed list of security issues in the Docker ecosystem. There exist a number of container scanning tools such as AppArmor [72], Anchore [61], Microscanner [206], Clair [70], and Cilium [51], which are used to detect security vulnerabilities in containers. Javed and Toor [115] investigated the quality of various container scanning tools based on coverage and accuracy. The results of their study demonstrate that the existing tools do not detect application package vulnerabilities and do not achieve high accuracy. Zerouali *et al.* [322] presented ConPan, an automated tool to monitor and analyze outdatedness, vulnerabilities, and bugs. This tool can be used as a CLI or integrated with other processes through its Python API. Table 3.12 summarizes the findings of existing studies related to container security approaches.

Table 3.12: Summary of studies on container security approaches and findings

| **Studies** | **Approaches Used** | **Findings** |
|---|---|---|
| [152] | Proposed DIVIDS (Docker Image Vulnerability | Result shows that DIVIDS provides a reliable docker environment. |

| | | |
|---|---|---|
| | Diagnostic System) | |
| [125] | Proposed DL-based algorithm that uses Restricted Boltzmann Machine (RBM) | The result shows that proposed approach protects containers and prevents security hacks of user application and workload in containerized environment. |
| [152] | Proposed DIVIDS (Docker Image Vulnerability Diagnostic System) | Result shows that DIVIDS provides a reliable docker environment. |
| [256] | Proposed DIVA (Docker Image Vulnerability Analysis) framework | The result shows that there exist more than 180 vulnerabilities in both official and community images. |
| [284] | Clair for static analysis, unsupervised algorithms for dynamic analysis | The static method detects 3 vulnerabilities, the dynamic approach detects 22 vulnerabilities and the combined solution of static and dynamic detection approaches detects 24 vulnerabilities out of 28 vulnerabilities. |
| [18] | Proposed Docker-sec methodology to protect the containers from outbreaks on container engines and servers. It also secures containers from zero-day vulnerabilities. | The result shows that Docker-sec performs better than existing approaches and provides a security mechanism for Docker. |
| [337] | Proposed *ZeroDVS*, a vulnerability detection and container image traceability mechanism | Result shows that *ZeroDVS* can be deployed in OS in order to trace the images and improve the security of the containers. |
| [235] | Introduced ML model to statically analyze container images to detect vulnerable Python libraries | This approach results in building a user interface to detect vulnerabilities in container images. |

**Methods to Resolve the Security Vulnerability Issue (RQ 4.2)**

Sultan *et al.* [265] presented a literature survey on security issues in containers and mechanisms to secure containers. They identified four use cases: protecting containers from applications, protecting the host from containers, inter-container protection, and protecting containers from the host. The first three use cases utilize software-based

solutions, while the last use case utilizes a hardware-based solution.

Gao *et al.* [78] identified container leakage channels that may expose details of host operating systems and containers. In this study, the authors launched an advanced power attack named synergistic power attack. In addition, they identified the main cause of information leakage and proposed a two-stage defense approach. The results show that the proposed method is effective and incurs minor performance overhead.

Andreou *et al.* [178] proposed Docker-sec, an automated mechanism to secure Docker containers via the enforcement of access policies with the help of AppArmor. By using Docker-sec, researchers can learn how it successfully defends containers from zero-day vulnerabilities with low performance overhead.

**Types of Attacks and Approaches used to Detect Security Attacks (RQ 4.3)**

Tomar *et al.* [280] proposed a threat model and presented a detailed classification of attacks based on various layers such as container, host, Docker engine, and application. Malware, DoS, ARP spoofing, and unauthorized access occur at the container layer. Poisoned images, outdated software, malware, and DoS take place at the allocation layer. Kernel exploits, cryptojacking, and malicious code injection occur at the Docker engine layer, while kernel exploits, tampering, and escape attacks occur at the host layer. They also provided a case study on denial-of-service (DoS) attacks in the Docker environment.

Huang *et al.* [105] presented an empirical study on existing Docker security mechanisms and the threats identified in Docker environments so far. They also proposed a threat detection mechanism and used a machine learning technique to predict unknown attacks in Docker images. Table 3.13 summarizes the findings related to approaches used to detect security attacks.

Table 3.13: Findings Related to Approaches to Detect Security Attacks

| Studies | Approaches Used | Findings |
|---|---|---|
| [155] | Proposed *SPEAKER* (Split Phase Execution of Application | Experimental results show that the proposed mechanism incurs less |

| | | |
|---|---|---|
| | Containers) | performance overhead and reduces system calls in order to secure containers. |
| [119] | Presented *CONSERVE* framework, a multi-domain and multi-dimensional framework to select appropriate container security mechanism | The result shows that the presented framework supports an efficient selection of container security monitoring techniques by saving time and effort in matching the techniques with the tasks at hand. |
| [168] | Proposed *CDL* (Classified Distributed Learning) framework that uses random forest algorithm for application classification and neural network algorithm for classified learning | The result shows that *CDL* outperforms simple anomaly detection methods as it reduces FPR from 12% to 0.24% and increases TPR from 45% to 74%. |
| [299] | Introduced ContainerGuard, a real time and noise resilient attack detection mechanism | The findings of this study depict that ContainerGuard performs well in detecting meltdown and spectre attacks with an AUC score of 0.9917 and incurs little running performance overhead in container-based big data platforms. |
| [328] | Proposed real time intrusion detection system that is based on OC-SVM (One Class-Support Vector Machine) algorithm | The result of this study demonstrates that OC-SVM detects security attacks with FPR ranging from 0.02 to 0.12 and TPR equal to 1. |
| [300] | Proposed DockerWatch to detect malware in container-based cloud environment. They also proposed CNN and LSTM based detection approach to maintain the tradeoff between high performance and fast real time malware detection. | The result of this study shows that proposed approach efficiently detects malwares with an improvement in runtime performance overhead in container-based cloud environment. |

### 3.3.5 RQ 5 Results

In hypervisor-based virtualization, there exists significant performance overhead because each VM runs on its own OS. Recently, a new prototype of virtualization technology has been introduced, which is named containerization, and it contains less performance overhead compared to VMs. In this, we will discuss performance evaluation techniques, benchmarks used, and monitoring tools used to evaluate container per-

formance. Bachiega *et al.* [24] presented a survey of studies involving containers and focused on the performance evaluation of containers. They categorized container performance evaluation techniques into three categories: analytical, simulation, and measurement. The results show that only a few studies used analytical and simulation techniques, whereas many studies have used benchmarks for performance evaluation, which comes under measurement techniques.

**Discussion**

- Studies related to various security vulnerabilities in containers, vulnerability detection approaches, approaches to mitigate the vulnerabilities, security attacks, and approaches to detect the security attacks are analyzed.
- There are some open-source tools such as anchore, apparmor, aqua trivy, clair, etc. to detect the vulnerabilities in containers. The authors introduced some frameworks such as DIVA, DIVIDS, DockerSec, ZeroDVS to detect and mitigate the security vulnerabilities.
- Malware, denial of service, malicious code injection, unauthorized access, cryptojacking, etc. are some of the attacks in containers. In the analyzed papers, CONSERVE, SPEAKER, ContainerGuard, CDL, etc. were used for detecting the attacks.
- In the analyzed papers, machine learning and deep learning algorithms were also used to detect the security vulnerabilities and security attacks in containers.

**Performance Evaluation Techniques (RQ 5.1)**

In the literature, many authors have introduced various frameworks and techniques to evaluate the performance of containers. Many studies have focused on the performance

comparison of hypervisor-based virtualization and containerization approaches.

Yoshimura *et al.* [320] introduced the ImageJockey framework to continuously evaluate the performance of container images. The experimental results demonstrate that there exist noticeable performance differences due to the deployed environments and features of container images.

Liu *et al.* [176] presented a performance evaluation methodology for industrial applications and then analyzed the results of the proposed methodology by applying it to a containerized edge cloud architecture. They also evaluated the processing power of containerizing an edge computing application in terms of implementing neural network and machine learning approaches for industry-based applications. The results show that the proposed approach can be used by edge-CC users as well as developers to obtain a client-perspective overview of system performance.

Manninen *et al.* [193] analyzed CPU usage, memory usage, and power consumption for various container platforms such as Docker, Balena, LXC, and Flatpak for use in Raspberry Pi-based monitoring and controlling devices. They used the mpstat script to evaluate CPU usage, the free script to evaluate memory usage, and for power consumption, they used the LabJack T7 device.

Plauth *et al.* [236] gave a comparative analysis of the performance of unikernels (Rumprun, OSv, MirageOS), containers (Docker, LXD), virtualization (KVM, Xen), native hardware, and combinations of these. The results of this study demonstrate that from the application throughput aspect, unikernels perform as well as or better than containers, and containers achieve shorter startup times compared to unikernels and virtual machines.

Abuabdo and Al-Sharif [4] evaluated the impact of containerization and virtualization techniques on multithreaded applications (written in C and Java). The results show that, in the case of macOS, a single-threaded C program performs better than a multithreaded Java program. In the case of Windows, single-threaded C programs give slightly slower results than single-threaded Java programs, but multithreaded C programs outperform multithreaded Java programs. The results for Ubuntu and Docker

are the same, as Docker images are deployed in a Ubuntu environment.

Kennedy Chengeta [46] studied the performance of traditional VM (Xen and KVM) and containers (Docker) and also analyzed the performance by combining two technologies—Docker and Kubernetes. The results show that Docker containers are faster and consume less memory than Xen and KVM. This study concludes that Docker with Kubernetes enhances performance and reduces delay as well as deployment time. Table 3.14 summarizes the existing approaches used to evaluate performance evaluation metrics in terms of resources utilized.

Table 3.14: Correlation between Technique and Performance Evaluation Metrics

| **Studies** | **Approaches Used** | **Performance Evaluation Metrics** |
|---|---|---|
| [176] | Presented a performance evaluation methodology for industrial applications and analyzed the results on containerized edge cloud architecture. | Round trip time, network bandwidth, CPU usage, and memory usage |
| [320] | Proposed ImageJockey framework for testing the performance of containers. This framework provides benchmark drivers, metric collection, and visualization tools. | CPU usage, throughput, elapsed time, response time, number of page faults, memory usage, etc |
| [193] | Used mpstat script to evaluate CPU usage, free script to evaluate memory usage and LabJack T7 for power consumption | CPU usage, memory usage, and power consumption |

**Benchmark Tools (RQ 5.2)**

In this section, we consider the findings related to the evaluation of container performance with the help of various benchmark tools. Several benchmark tools have been used in the literature to evaluate container performance, such as LINPACK, Y-CRUNCHER, STREAM, NETPERF, BONNIE++, and SYSBENCH, among others.

Psutil is a Python-written benchmarking library that is used to evaluate CPU utilization, memory utilization, disk utilization, network utilization, and sensor utilization. It is mainly useful for monitoring the system, profiling, limiting process resources, and

managing running processes.

Ann Mary Joy [120] compared containers and VMs in terms of application performance and scalability. They used JMeter to evaluate application performance and scalability. The results of this study demonstrate that, in terms of scalability and performance, containers outperform virtual machines. Containers can be used for deploying applications to reduce resource overhead because of their better resource utilization and scalability.

Benchmark tools are broadly classified into four categories: CPU-intensive benchmarks, memory-intensive benchmarks, disk I/O-intensive benchmarks, and network I/O-intensive benchmarks.

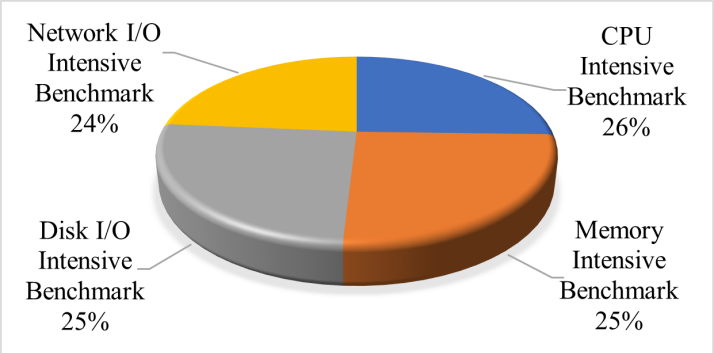



Figure 3.10: Benchmark Tools Used

Figure 3.10 shows the pie chart of benchmark tools used for performance evaluation of containers. This figure illustrates that 26% of the studies used CPU-intensive benchmark tools, 25% of the studies used memory-intensive benchmark tools, 25% of the studies used disk I/O-intensive benchmark tools, and 24% of the studies used network I/O-intensive benchmark tools for performance evaluation. Table 3.15 describes the correlation between benchmark tools and performance evaluation metrics.

Table 3.15: Correlation between Benchmark Tools and Evaluation Metrics

| **Studies** | **Benchmark Tools** | **Performance Evaluation Metrics** |
|---|---|---|
| [213], [141], [234], [116], [196], [229], [197], [28], [212], | Y-cruncher, Linpack, Nbench, Noploop, Pbzip2, Geekbench, | CPU usage |

| | | |
|---|---|---|
| [200], [238], [5], [37], [248] | Sysbench, Spec CPU 2006 | |
| [213], [141], [116], [196], [229], [197], [248], [164], [5], [234], [200], [212], [238], [25] | Stream, RAM Speed/SMP | Memory usage |
| [217], [213], [141], [116], [164], [28], [234], [37], [212], [238], [196], [197], [25], [248] | Bonnie++, Sysbench, Iozone, Fio | Disk I/O usage |
| [213], [141], [116], [196], [229], [197], [248], [234], [154], [164], [25], [5], [212] | Netperf, Iperf, Nuttcp | Network bandwidth usage |

**CPU Intensive Benchmarks (RQ 5.2.1)**

In this category, Y-Cruncher, LINPACK, GEEKBENCH, and SYSBENCH are widely used CPU intensive benchmark tools. Table 3.16 shows various CPU intensive benchmark tools used in the existing studies.

Table 3.16: CPU Intensive Benchmarks

| **Benchmark Name** | **Description** | **Literature Studies** | **Percentage** |
|---|---|---|---|
| NBENCH [225] | NBench stands for Native mode Benchmark. It is a single-threaded tool used to measure CPU, FPU, and memory speed, and it produces three indexes: floating point index, integer index, and memory index. | [213] | 4% |
| Y-CRUNCHER [315] | Y-cruncher is a multi-threaded and scalable benchmark tool for multi-core systems. It computes Pi and other constants to trillions of digits and is executed as a stress testing tool for the CPU. It produces various outcomes such as total computation time, execution time, | [213], [141], [234] | 12% |

| | | | |
|---|---|---|---|
| | and multicore efficiency. | | |
| NOPLOOP [278] | The "noploop" is a very simple CPU benchmarking tool: *"It is a procedure for measuring CPU clock speed using an unrolled No-Operation (NOP) loop. It's intended to be simple, minimizing variation caused by cache misses, stall cycles, and branch misprediction."* | [213] | 4% |
| LINPACK [169] | Linpack is an HPC benchmarking tool used to solve simple linear equations. It performs lower upper decomposition of numerical analysis with partial pivoting. It describes how fast a computer will perform when solving real problems. It generates the outputs in terms of MegaFLOPS (Floating point operations per second). | [213], [141], [116], [196], [229], [197], [28], [212] | 32% |
| GEEKBENCH [79] | Geekbench is a cross platform that analyzes both single core and multi core architectures to compute processor power. Multi core architecture supports the concept of hyperthreading but single core architecture does not support hyperthreading as it uses only one CPU core. It produces various outcomes such as integer performance, memory performance, and floating-point performance. | [141], [28], [200] | 12% |
| PBZIP2 [232] | PBZIP2 is a parallel version of BZIP2 data compression technique which uses pthreads for parallel file compression. It measures the time needed to compress a file. This tool runs faster on multiple CPUs as it firstly divides the file into pieces and then compresses each piece individually. | [141] | 4% |

| | | | |
|---|---|---|---|
| SYSBENCH [266] | Sysbench is an open-source benchmark tool used to benchmark Linux. It calculates CPU utilization, memory utilization, disk I/O and database performance. In the case of CPU benchmark, Sysbench evaluates the time required to perform the maximum prime number. | [238], [229], [28], [5], [234], [37], [212] | 28% |
| SPEC CPU 2006 [263] | SPEC CPU 2006 is introduced by SPEC (Standard Performance Evaluation Corporation) and stressing memory subsystem, system processor, and compiler. Currently, this benchmark is replaced by SPEC CPU 2017 benchmark suite. It evaluates speed and throughput. | [248] | 4% |

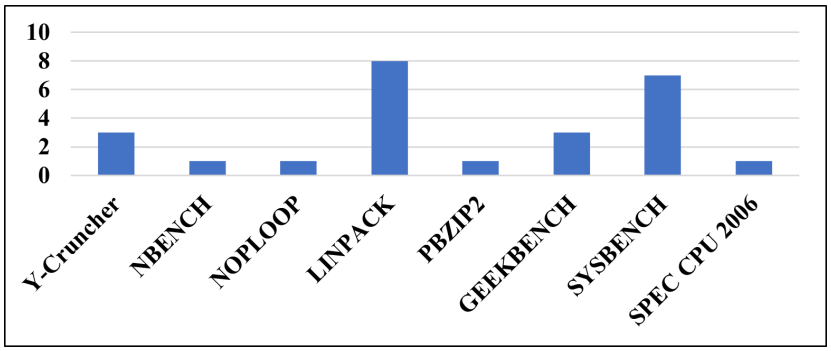



Figure 3.11: CPU Intensive Benchmark Tools Used

Figure 3.11 shows bar graph of CPU intensive benchmark tools used in the performance evaluation of containers. This figure describes that LINPACK is the most used CPU intensive benchmarks tool with a percentage of 32% and NBENCH, NOPLOOP, PBZIP2, and SPEC CPU 2006 are the least used benchmark tools.

**Memory Intensive Benchmarks (RQ 5.2.2)**

There exist various benchmark tools to measure memory performance such as STREAM, Graph500, FEniCS, RAM speed/SMP. Graph500 evaluates TEPS (Traversed Edge Per Second) and executes graph data structure to obtain BFS (Breadth First Search) in the data construction and data generation phase. FEniCS solves the Poisson equation and

is used in distributed memory parallelism. Table 3.17 shows various memory intensive benchmark tools used in the existing studies.

Table 3.17: Memory Intensive Benchmarks

| **Benchmark Name** | **Description** | **Literature Studies** | **Percentage** |
|---|---|---|---|
| STREAM [264] | Stream is an industry standardized memory intensive benchmark that is used to measure sustained memory bandwidth in MB/s. It is also a part of HPC (High Performance Computing) challenge benchmark suite. Stream utilizes four kernels: copy, add, triad and scale and accesses memory throughput. | [213], [141], [116], [196], [229], [197], [248], [164], [5], [234], [200], [212] | 86% |
| RAM speed/SMP [84] | RAM speed/SMP (shared memory multiprocessing or symmetric multiprocessing) is a cache and memory benchmark tool used to measure system memory (both RAM and cache) bandwidth for multiprocessor machines. It consists of two benchmarks (mark and mem): 1. INTmark, FLOATmark, MMXmark, SSEmark and 2. INTmem, FLOATmem, MMXmem, SSEmem. | [238], [25] | 14% |

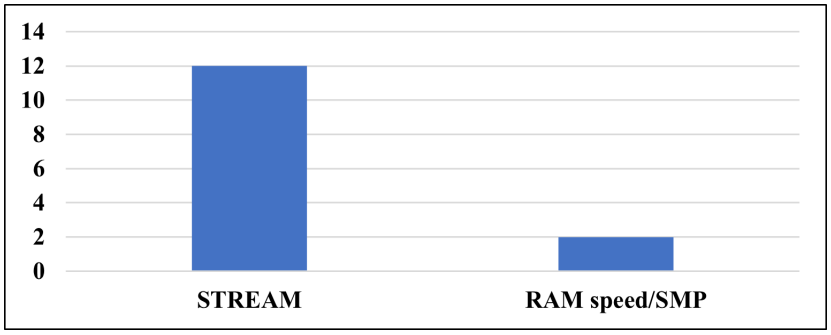



Figure 3.12: Memory Intensive Benchmark Tools Used

Figure 3.12 shows bar graph of memory intensive benchmark tools used in the performance evaluation of containers. This figure describes that STREAM is the most used

memory intensive benchmarks tool with a percentage of 86% as compared to RAM speed/SMP tool.

#### Disk I/O Intensive Benchmarks (RQ 5.2.3)

These types of benchmark tools are used to test disk I/O operations. Tools such as Bonnie++, Iozone, Sysbench, Fio, Dbench, dd test, etc are various disk intensive benchmark tools. Dbench [65] is an emulation of the file system load function of the Netbench benchmark. We created 100 parallel client processes of Dbench to generate load for the file system. Every process begins at the same time and executes the same workload. To test the read and write capacity of the disk system, DD test benchmark is used. Table 3.18 shows various disk I/O intensive benchmark tools used in the existing studies.

Table 3.18: Disk I/O Intensive Benchmarks

| **Benchmark Name** | **Description** | **Literature Studies** | **Percentage** |
|---|---|---|---|
| Bonnie++ [35] | Bonnie++ is a benchmark suite that is aimed at performing several simple tests of hard drive and file system performance. This tool is used to evaluate the performance of the system based on read/write speed, read/write capability of a system, and also benchmark the number of read-/write operations that can be performed per second. | [217], [213], [141], [116], [164], [28], [234] | 37% |
| SYSBENCH [266] | Sysbench is an open-source benchmark tool used to benchmark Linux. It calculates CPU utilization, memory utilization, disk I/O and database performance. To measure file I/O performance, we first need to create a test file that is much bigger than your RAM (150GB is a good value). | [141], [28], [234], [37], [212] | 26% |
| IOZONE [109] | IOZONE is a benchmark tool for measuring file I/O performance. To measure file I/O performance, this tool performs various operations such as read, re-read, read, | [238], [196], [197], [25] | 21% |

| | | | |
|---|---|---|---|
| | write, re-write, etc. | | |
| FIO [1] | FIO (Flexible Input Output Tester) is a benchmark tool used to measure disk I/O performance by evaluating IOPS (Input/Output Operations Per Second). It has the flexibility to select different IO sizes, sequential or random reads, and writes. It spawns worker threads or processes to perform the specified I/O operations. | [248], [28], [212] | 16% |

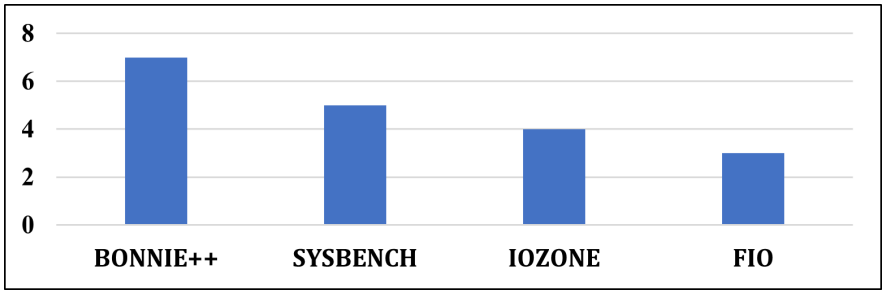


Figure 3.13: Disk I/O Intensive Benchmark Tools Used

Figure 3.13 shows bar graph of disk i/o intensive benchmark tools used in the performance evaluation of containers. This figure describes that BONNIE++ is the most used disk i/o intensive benchmarks tool with a percentage of 37% and FIO is the least used disk i/o intensive benchmark tool.

### Network I/O Intensive Benchmarks (RQ 5.2.4)

There exist various benchmark tools to measure network performance such as Netperf, Iperf, Netpipe, Nuttcp, Iperf3. NetPIPE [227] stands for Network Protocol Independent Performance Evaluator. Its protocol-independent feature shows overhead associated with the protocol layer. It encapsulates the advantages of ttcp and netperf. Iperf [110] is a network intensive benchmark tool used to analyze the network performance in terms of throughput, speed, and bandwidth. This tool creates TCP and UDP streams to measure throughput. It follows client server model to measure throughput between two systems. Table 3.19 shows various network I/O intensive benchmark tools used in the existing studies.

Table 3.19: Network I/O Intensive Benchmarks

| **Benchmark Name** | **Description** | **Literature Studies** | **Percentage** |
|---|---|---|---|
| NETPERF [226] | This tool measures network bandwidth between two hosts on a network. Netperf is designed on a client-server model. There are two executable components in Netperf: netperf (network performance benchmark client) and netserver (network performance benchmark server). It measures request/response performance using TCP, UDP, and SCTP. | [213], [141], [116], [196], [229], [197], [248], [234], [154] | 53% |
| IPERF [110] | Iperf is a network intensive benchmark tool used to analyze network performance in terms of throughput, speed, and bandwidth. This tool creates TCP and UDP streams to measure throughput. It follows a client-server model to measure throughput between two systems. | [141], [229], [164], [25], [5], [212] | 35% |
| NUTTCP [301] | Nuttcp is a network performance measurement tool that is used by network and system managers. It evaluates throughput in Mbps and also provides other information such as user and system wall clock time, transmitter and receiver CPU utilization, and loss percentage. | [229], [234] | 12% |

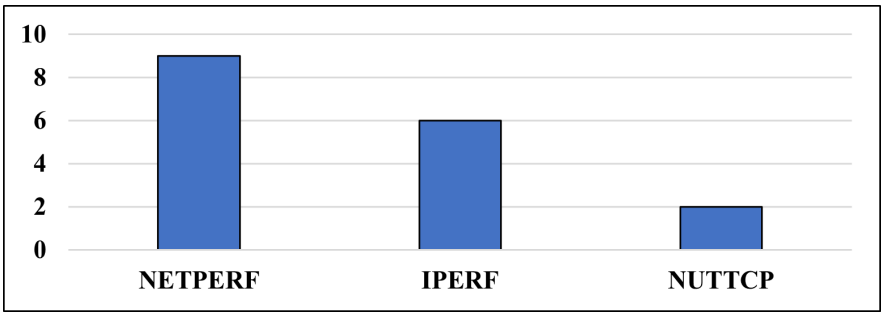



Figure 3.14: Network I/O Intensive benchmark tools

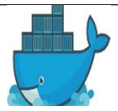
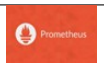
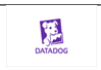
Figure 3.14 shows bar graph of network i/o intensive benchmark tools used in the performance evaluation of containers. This figure describes that NETPERF is the most used network i/o intensive benchmarks tool with a percentage of 53% and NUTTCP is the least used network i/o intensive benchmark tool.

### Monitoring Tools (RQ 5.3)

Container monitoring tools track the operation of containerized applications. Monitoring helps in detecting and resolving the issues at an early stage and ensures optimal system performance all the time. Container monitoring tools assess metrics such as CPU utilization, memory utilization, disk i/o utilization, network bandwidth, CPU limit, and memory limit. [214] presented ConMon tool to monitor the network performance of container-based applications. It is a distributed automated monitoring system that automatically identifies those containers that communicate with each other and analyze the network performance between these containers. The result of this study depicts that ConMon system introduces low overhead on resource utilization and negligible overhead on network performance but it can affect throughput and latency of monitored container-based applications. DEEP-mon, a monitoring tool to measure power consumption was proposed by [36]. It is a dynamic and energy-efficient power consumption tool that uses 6 benchmarks EP, MG, CG, pts/apache, pts/nginx, pts/-postmark. The experimental result shows that DEEP-mon has a negligible overhead on power consumption and performance. CMonitor, a monitoring and alarming tool for containers proposed by [117]. This tool provides integrated monitoring services, rich visualization to record the data, alarming mechanisms in case of detecting abnormal behavior in containers, and a global topology view. The experimental evaluation depicts that CMonitor can accurately collect CPU utilization, memory utilization, and block I/O metrics. We have presented a summative study of various container performance monitoring tools in table 3.20.

Table 3.20: Container Performance Monitoring Tools

| Tool name | Logo | Developed by | Visualization tools | Performance metrics | Supported OS | License | Fee | Send Alert | Target platform |
|---|---|---|---|---|---|---|---|---|---|
| Sensu [251] | Sensu by sumologic | Sensu | Grafana | CPU, Memory and network I/O usage | Linux, Windows | Open source, Commercial | Paid | Yes | Docker, LXC,rkt |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Sematext [58] |  | Sematext | Kibana, Sematext Logs UI | CPU usage, Disk I/O usage, Memory usage, and Network IO stats | Linux | Open source | Paid | Yes | Docker, rkt |
| Docker stats [71] |  | Docker Inc. | No | CPU and memory usage, memory limit, networking information and disk utilization | Linux | Open source | Free | No | Docker |
| Sysdig [54] |  | Sysdig | Sysdig Monitor | CPU, Memory and network I/O usage | Linux, Windows | Open source, Commercial | Paid | Yes | Docker, LXC,rkt |
| Prometheus [231] |  | Sound Cloud | Node exporter, Grafana, Prometheus UI | CPU and memory usage and network bandwidth | Linux, Windows, Darwin | Open Source | Free | Yes | Docker |
| cAdvisor [211] |  | Google | Grafana, cAdvisor Web UI | CPU and memory usage, network i/o and disk space utilization | Linux | Open source | Free | No | Docker |
| Datadog [55] |  | Olivier Pomel, Alexis Lê-Quôc | Dashboard | CPU and disk usage | Window, Linux, Android, iPhone and MacOS | CommercialPaid | | Yes | Docker, Kubernetes |
| Scout [59] |  | Scout | Dashboard | CPU and memory usage | Linux | Open source, Commercial | Paid | Yes | Docker, rkt |
| Dynatrace [56] |  | Dynatrace | Smartscape | CPU usage, CPU limit, memory limit and memory usage | Linux, Unix, Windows and z/OS | Open source | Paid | Yes | Docker, Kubernetes, Mesos, Open shift |
| Appoptics [57] |  | Solar winds | Dashboard | CPU utilization, memory limits and utilization, network utilization, and error metrics | Linux, MacOS, iPhone | Open source, Commercial | Paid | Yes | Docker, Kubernetes |

**Discussion**

- Our analysis reveals the introduction of various frameworks for assessing container performance along with the utilization of benchmark tools and monitoring tools for performance evaluation of containers.
- The literature offers a list of various CPU intensive (Ycruncher, NBENCH, NOPLOOP, LINPACK, etc.), memory intensive (STREAM, RAM speed/SMP), disk I/O intensive (Bonnie++, Sysbench, IOZONE, etc.), and network I/O intensive (Netperf, Iperf, Nuttcp, etc.) benchmark tools.
- Notably, LINPACK serves as a widely used CPU intensive benchmark tool, STREAM serves as a widely used memory intensive benchmark tool, BONNIE++ serves as a widely used disk I/O intensive benchmark tool, and NETPERF serves as a widely used network I/O intensive benchmark tool.
- Various organizations have also introduced a list of container performance monitoring tools such as cAdvisor, Prometheus, Grafana, Docker stats, Sematext, Dynatrace, etc.

## 3.4 Conclusion and Future Guidelines

In this paper, the literature on the maintenance of software containers is consolidated by defining RQ framework based on container scheduling techniques, container performance evaluation, container security measures, container performance monitoring tools, image tagging approaches, and image detection approaches. Firstly, we have collected 208 studies from various databases such as IEEE Xplore, Google Scholar, ACM Digital Library, Springer link, Science direct and then we have selected 145 primary studies based on quality assessment criteria and inclusion and exclusion criteria. We have also read 33 hyperlinks to collect the data related to container performance evaluation benchmark tools such as CPU intensive benchmarks tools, memory inten-

sive benchmark tools, disk i/o intensive benchmark tools, and network i/o intensive benchmark tools and container performance monitoring tools. From the summative analysis of selected primary studies on containers, it is found that

- 42% of studies are related to container scheduling techniques, 24% of studies are related to container performance evaluation, 22% studies are related to container security measures, and 12% studies are related to container image detection process.
- From the perspective of container scheduling algorithms, it is found that 33% studies have used heuristic approaches, 32% studies are related to meta heuristic approaches, 18% studies have used mathematical modeling approaches, and 17% studies have used ML approaches.
- In case of heuristic approaches, resource utilization is the most considered performance objective and network transmission is the least considered performance objective. In case of meta heuristic approaches, resource utilization is the most considered performance objective and scalability is the least considered performance objective. In case of mathematical modeling approaches, resource utilization is the most considered performance objective and availability is the least considered performance objective. In case of ML approaches, load balancing is the most considered performance objective and makespan is the least considered performance objective.
- Authors have identified various security issues and proposed various defense methods and approaches to detect and resolve security vulnerabilities in containers. Authors have also used various open-source vulnerability scanning tools to detect potential threats and security vulnerabilities in containers.
- Authors have introduced various frameworks to evaluate the performance of containers and have also used benchmark tools for performance evaluation. There exists various CPU intensive, memory intensive, disk i/o intensive, and network i/o intensive benchmark tools in the literature. LINPACK is widely used CPU

intensive benchmark tool, STREAM is widely used memory intensive benchmark tool, BONNIE++ is widely used disk i/o intensive benchmark tool, and NETPERF is widely used network i/o intensive benchmark tool.

Since we expect this research area to continue to grow in the future, we hope that our analysis will become useful in developing new approaches for maintaining software containers The following are the future guidelines for researchers to carry out research on maintenance of software containers:

- There are few studies related to container image detection, image tagging approaches, and container security measures. So, there is a need to explore container image detection process and propose new image tagging approaches, new smell detection approaches, and new approaches to detect the necessary images to be included in container.
- There is also a need to explore container security measures and develop new approaches to detect and mitigate security threats as containers are less secure.
- There are few studies that have used ML and mathematical modeling approaches in container scheduling. So, in future, we can propose new container scheduling techniques with the help of ML techniques and mathematical modeling techniques.
- As resource utilization is the most considered performance objective in almost all the scheduling approaches so we can analyze the results on the basis of other performance objectives also such as scalability, throughput, network transmission, cost etc.

# Chapter 4

# WORKLOAD FORECASTING

## 4.1 Introduction

CC has become a key paradigm in the field of information technology, providing scalable and on demand resource allocation to satisfy the dynamic needs of modern applications. Despite its benefits, CC faces some challenges; the ineffective resource allocation for dynamic workloads is one of them. Predicting workload in CC environment is an important part of auto scaling and resource management.

Accurate workload prediction enables cloud providers to dynamically allocate resources, optimize task scheduling, and avoid both over-provisioning and under-provisioning of systems. A typical workload prediction process operates as an iterative cycle. In this cycle, a predictive model generates forecasts based on historical workload data. These predictions inform cloud managers or automated algorithms when implementing resource allocation strategies, performing elastic scaling, or balancing workloads across available resources. As workload patterns evolve over time, the accumulated historical data is fed back to update and refine the predictive model.

Forecasting workload is a complex task due to the dynamic nature of workload patterns, which often exhibit sharp spikes and irregular fluctuations. Consequently, this problem has been approached as a time series analysis challenge, with various meth-

ods applied including statistical techniques [146], evolutionary learning [147][149], traditional ML approaches, and more recently, advanced DL architectures like LSTM [144], autoencoders [330], deep neural network [314], hybrid learning [31][45], ensemble learning [259][148], and quantum learning [257]. The categorization and taxonomy of the commendable and recent cloud workload prediction models based on ML algorithms are depicted in the figure, where they are classified according to their conceptual and operational characteristics. ML and DL has demonstrated notable advancements in various fields such as image recognition and speech recognition, offering novel perspectives for predicting workload in CC environment but these techniques do not perform well on distributed data.

However, workload data is typically large-scale, heterogeneous, and distributed across multiple data centers, posing significant challenges for centralized training of ML and DL models related to privacy, data transfer costs, and scalability. To overcome these obstacles, FL has emerged as a privacy-preserving distributed ML paradigm that allows multiple clients—such as cloud nodes or edge devices—to collaboratively train a model without exchanging raw data. Instead of moving data to a centralized server, FL brings the model to the data, thus maintaining privacy while reducing communication overhead.

This chapter presents an FL-based workload prediction framework using CNN as the base model, evaluated on four large-scale real-world datasets — Alibaba 2018, Alibaba 2020, GCT 2011, and GCT 2019. A comparative analysis is performed against conventional CNN and LSTM models. Experimental results demonstrate that the proposed FL-CNN framework achieves superior accuracy and generalization, ensuring privacy preservation and scalability in distributed cloud environments.

The remaining sections are organized in the following manner: The simulation approach that is being proposed is outlined in section 2 including dataset description, techniques used, and performance evaluation measure used to evaluate the efficacy of the model. Afterwards, in section 3, the experimental findings and the outcome of the proposed approach compared with other DL approaches such as CNN and LSTM are

discussed. Lastly, Section 4 provides a description of the conclusion and future work.

## 4.2 Research Methodology

This section details the procedures and methodologies implemented in the empirical investigation. It presents a systematic overview of the approach taken to design, implement, and evaluate the predictive models developed within this chapter. Key components include a description of the datasets utilized, preprocessing methods employed to prepare the data, the classification algorithms applied, and the performance metrics leveraged to assess the effectiveness and accuracy of the models.

### 4.2.1 Dataset

To evaluate the proposed FL framework under realistic and diverse conditions, this study utilizes four large-scale, real-world cloud workload datasets: Alibaba v2018 [118], Alibaba 2020 [302], GCT 2011 [243], and GCT 2019 [305]. These publicly available datasets are established benchmarks, representing a wide spectrum of temporal resolutions, workload heterogeneity, and distributed infrastructure scales. Table shows the statistical characteristics of each dataset.

The Alibaba v2018 dataset captures eight days of CPU and memory usage from around 4,000 machines, including both interactive and batch job types, offering high-frequency resource measurements ideal for analyzing short-term workload spikes and prediction responsiveness. Alibaba 2020 extends this, providing two months of CPU, GPU, and memory traces from over 1,800 machines and 6,500 GPUs, enabling long-term analysis across daily and weekly cycles in environments increasingly reliant on GPU-intensive tasks.

GCT 2011 contains 29 days of detailed CPU, memory, and disk usage records from more than 12,500 machines, including job and task scheduling events and failure logs. This dataset exposes both diurnal workload patterns and operational anomalies, supporting evaluation of the model's handling of dynamic, failure-prone environments.

The updated GCT 2019 covers 29 days across eight cluster cells, each with approximately 10,000 machines, and includes advanced workload management metrics such as task latency and scheduling delays, reflecting modern heterogeneous cloud deployments.

Collectively, these datasets represent a rich variety of resource utilization behaviors and workload distributions, providing a rigorous testbed for assessing the adaptability, scalability, and privacy-preservation capabilities of the proposed FL framework in non-IID, real-world cloud environments.

Table 4.1: Statistical characteristics of dataset

| Dataset | Duration | No. of Machines | Sampling Rate | Resources Used | Data Size (GB) |
|---|---|---|---|---|---|
| Alibaba 2018 | 8 days | 4000 | Seconds | CPU, memory | 13.2 |
| Alibaba 2020 | 2 months | 1800 | Seconds | CPU, GPU, memory | 25.7 |
| GCT 2011 | 29 days | 12500 | Minutes | CPU, memory | 17.9 |
| GCT 2019 | 29 days | 10000 | Minutes | CPU, memory, disk | 20.4 |

### 4.2.2 Data Preprocessing

Similarly to other works in the literature [99] [145] [170][247], the raw workload traces undergo a multi-step preprocessing procedure to ensure uniformity and improve model convergence. Each dataset is preprocessed independently to account for differences in data detail and storage structure. Google BigQuery was utilized for efficient querying and handling of the large-scale GCT 2019 dataset ($\tilde{2}$.4 TiB compressed).

For each cluster, we construct time-series datasets representing the average CPU and average memory utilization across all machines at 5-minute intervals. Missing records are neglected for simplicity, as their frequency is minimal relative to the overall data size. In cases where program tasks execute only partially within a 5-minute window,

the average resource utilization is multiplied by a weight proportional to the fraction of the interval during which the task is active. This ensures that each data point accurately reflects the effective workload contribution of ongoing tasks.

After temporal aggregation, the data is scaled to the range [0, 1] using the Min–Max normalization technique, which helps accelerate model convergence and maintain numerical stability during training.

This preprocessing ensures consistent temporal granularity, balanced feature scaling, and uniform representation of workload behavior, providing a clean and structured input predictive model developed in this chapter.

### 4.2.3 Techniques

In this study, FL is used for predicting workload in CC environment. From figure 4.1, firstly, base DL model CNN is trained using server data to construct the central server model. The central server model is subsequently distributed to each client, where clients utilize their local

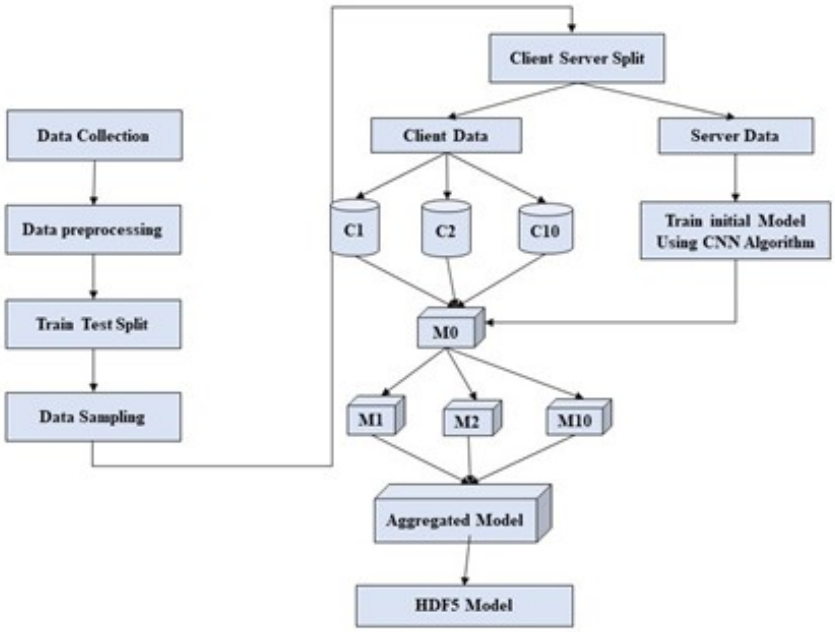



Figure 4.1: Research Framework

data on their respective devices to train the central server model and construct a client model. The weights of all client models are then aggregated using Federated Averaging (FedAvg) method to reconstruct a central server model, which is finally sent back to the server. While the conventional ML techniques involve moving data to the model, FL involves moving the model to the data. FL is especially useful in scenarios where concerns regarding privacy of data arises from data sharing, and when the data is either too extensive or too sensitive to be centralized. The detailed description of FL is given in section of chapter 2.

The CNN consists of eight 1D convolutional layers with varying dilation rates to effectively capture multi-scale temporal dependencies. The Adam optimizer is employed with ReLU activation in hidden layers and sigmoid activation in the output layer. Dropout and batch normalization layers are included to prevent overfitting and stabilize learning.

Table 4.2: Hyperparameter Details

| Components of the Proposed Framework | Hyperparameters |
|---|---|
| Iterations | 20 |
| Clients | 10 |
| Verbose | 0 |
| Epochs | 50 |
| Batch size | 64 |
| Activation Function | ReLu, and Sigmoid (At output layer) |
| Optimizer | Adam |
| Loss function | Binary cross entropy |
| Number of Convolution 1D layers | 8 |
| Regularization Method | Dropout, BatchNormalization |

### 4.2.4 Performance Analysis

The forecasting accuracy of the proposed model is assessed using widely recognized time-series performance metrics MAE, MSE, RMSE, MAPE, and $R^2$. The detailed

description of these metrics is given in section of chapter 2. Lower MAE, MSE, RMSE, and MAPE values indicate higher predictive accuracy, while $R^2$ values closer to 1 reflect a stronger correlation between predicted and actual workloads.

## 4.3 Experimental Result

The proposed model is evaluated on Alibaba cluster trace v2018, Alibaba cluster 2020, GCT 2011, and GCT 2019 datasets. The results of the proposed model are analyzed based on performance metrics. Note that MAE, MSE, MAPE, RMSE, and R2 measures are used to assess the predictive accuracy of the model. The performance of the proposed model is compared with CNN and LSTM.

Table 4.3 shows the experimental findings of FL model in terms of MAE, MSE, RMSE, MAPE, and R2. According to table 2, for Alibaba cluster trace 2020 error rate (0.0005) is less among all the selected dataset and R2 value (0.984) is high as compared to other datasets. Furthermore, we assess the efficacy of the proposed model by contrasting its prediction accuracy with CNN and LSTM models.

Table 4.3: Experimental Results

| | Alibaba 2018 | Alibaba 2020 | GCT 2011 | GCT 2019 |
|---|---|---|---|---|
| MAE | 0.07 | 0.016 | 0.02 | 0.023 |
| MSE | 0.006 | 0.0005 | 0.004 | 0.0009 |
| RMSE | 0.077 | 0.0237 | 0.06 | 0.031 |
| MAPE | 7.37 | 1.66 | 4.26 | 2.385 |
| $R^2$ | 0.337 | 0.984 | 0.923 | 0.979 |

Figure 4.2 shows the difference between actual CPU usage rate and predicted CPU usage rates of the proposed model for Alibaba cluster trace v2018, Alibaba cluster trace 2020, GCT 2011, and GCT 2019 datasets. In figure 2, x-axis shows the time series data and y-axis shows CPU usage rates. From the figures, it can be concluded that except for google cloud trace 2019 dataset, the actual values and predicted values are almost similar for all other datasets. In other terms, we can say that error rate is very less in case of almost all the datasets for the proposed model.

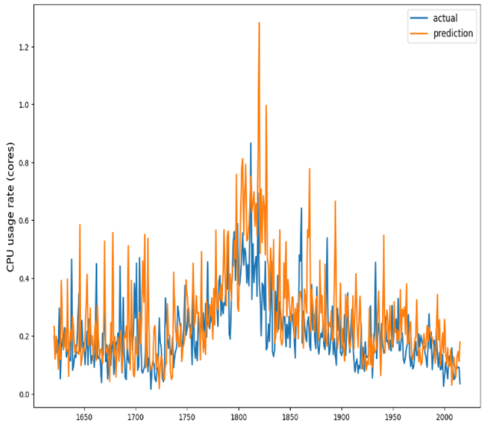



Fig (a) Alibaba cluster trace 2011 dataset

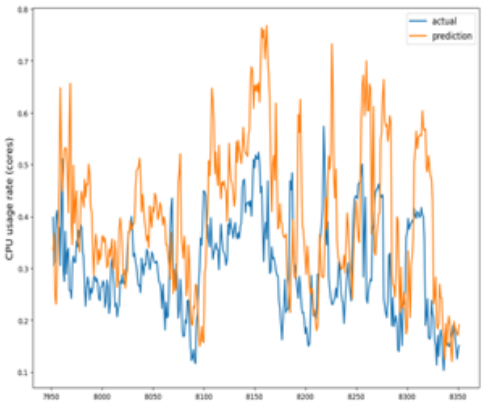



Fig (b) GCT 2011 dataset

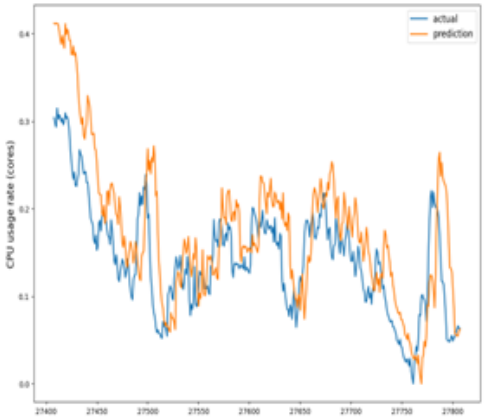



Fig (c) Alibaba cluster trace 2020 dataset




Fig (d) GCT 2019 dataset

Figure 4.2: Comparison between actual CPU usage rates and predicted CPU usage rates of the proposed model for alibaba cluster trace v2018, alibaba cluster trace 2020, GCT 2011, and GCT 2019 datasets.

For benchmarking, traditional CNN and LSTM models are trained on centralized data subsets extracted from the same datasets. These serve as baselines to measure predictive accuracy and efficacy against the proposed federated learning model. Figure 4.3 shows the comparative analysis of proposed model with CNN and LSTM for Alibaba cluster 2020 and google cloud 2019 dataset in terms of MAE, MSE, MAPE, RMSE, and R2 measures. It can be concluded that the proposed model out-performs CNN and LSTM and decreases the MAPE, MSE, MAE, and RMSE values while increasing R2 value.

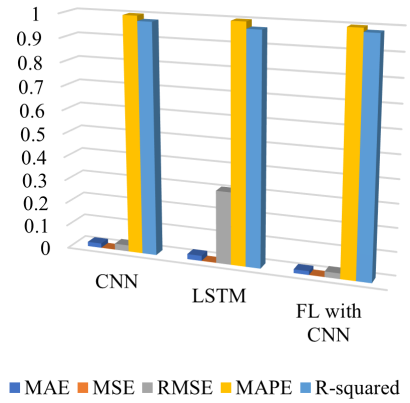



Fig (a) Comparison between CNN, LSTM, and FL with CNN for alibaba cluster trace 2020 dataset

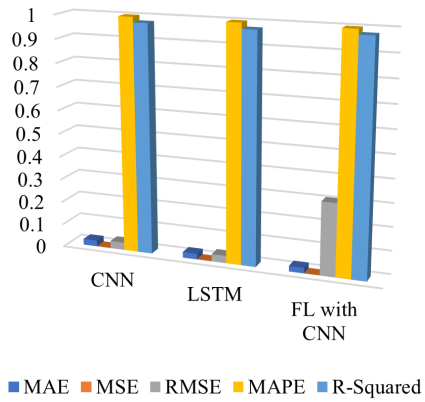



Fig (b) Comparison between CNN, LSTM, and FL with CNN for google cloud trace 2019 dataset

Figure 4.3: Comparison of proposed approach with CNN and LSTM

## 4.4 Conclusion

This chapter introduced a FL based framework for cloud WF that effectively addresses the challenges of scalability, privacy, and data heterogeneity characteristic of distributed cloud infrastructures. The study leveraged four expansive real-world datasets—Alibaba 2018, Alibaba 2020, GCT 2011, and GCT 2019—to rigorously evaluate the predictive efficacy of the proposed FL model, which employs a CNN as the base learner.

Comprehensive experimental analyses revealed that the FL-based CNN substantially outperforms traditional models, such as standalone CNN and LSTM, across multiple datasets. The FL-CNN showed consistently superior forecasting accuracy, evidenced by lower values of MAE, MSE, RMSE, and MAPE, alongside higher $R^2$ value. Particularly notable was the performance on the Alibaba 2020 dataset, where the model achieved an $R^2$ of 0.984 and an exceptionally low error rate of 0.0005, underscoring its robust generalizability and stability.

These findings have practical implications for cloud resource management. It empowers service providers to dynamically adjust resources, preempt SLA violations, and

optimize infrastructure usage. Moreover, by adopting the FL paradigm, the proposed model inherently preserves data privacy, rendering it especially suitable for scenarios involving multi-tenant and hybrid cloud architectures where data confidentiality and collaborative operations across distinct entities are paramount.

Overall, the findings of this study indicate that FL constitutes a robust, secure, and scalable paradigm for real-world cloud WF, supporting intelligent and autonomous resource management across distributed data centers. The demonstrated success of the FL-based workload prediction model establishes a strong foundation for the subsequent research phase, which extends to CFP. While WF is primarily focused on anticipating fluctuations in resource demand to enable efficient allocation, failure prediction expands the model's utility toward proactive system health management, identifying potential faults before they manifest. The next chapter will explore this critical aspect in detail, integrating advanced temporal DL and optimization techniques to further enhance the resilience and reliability of CC environments.

# Chapter 5

# DEVELOPMENT OF CLOUD FAILURE PREDICTION MODELS

## 5.1 Introduction

CC is a platform that consists of interconnected clusters of virtual and physical infrastructure. Cloud Service Providers (CSPs) offer customers online access to a large pool of computing resources for hosting and executing their application tasks. Customers are billed based on the resources their tasks consume, using a pay-as-you-go model. This pricing strategy attracts many customers to move their tasks to the cloud. According to Business Fortune Insight, the North American market has invested around $78.28 billion in cloud services. The CC market is expected to experience steady growth, with its value rising from $219 billion in 2020 to an anticipated $791.48 billion by 2028. As the demand for cloud-based task execution increases, ensuring task availability becomes important for meeting customer expectations. Task availability refers to the percentage of time that tasks remain accessible to customers [112]. CSPs are responsible for ensuring high levels of Quality of Service (QoS) to guarantee task availability.

Like any other computing system, CC is suspectable to failures. A CC system fails

when it cannot execute its predefined functions due to hardware malfunctions or unforeseen software issues. The complexity of the computing system directly correlates with the likelihood of failures occurring. Failures in CC are typically classified into two categories: architecture-based and occurrence-based failures [83]. Table 5.1 summarizes the various types of cloud failures and causes of cloud failures. Since the focus of this study revolves around the job and task termination status, this study aligns with service failures, primarily driven by scheduling issues.

Table 5.1: Summary of Categories, Types, and Causes of Cloud Failures

| **Categories** | **Types of Cloud Failures** | **Causes** |
|---|---|---|
| Architecture based | Service Failure | Software Failure (cyberattacks, rebooting at any unplanned time, etc.) |
| | | Scheduling Failure (service timeout) |
| | Resource Failure | Hardware Failure (memory issue, system breakdown, power outages, etc.) |
| Occurrence based | Correlated Failure | Spatial or temporal correlation of 2 failures |
| | Independent Failure | Denser System Packing |
| | | Human Error |
| | | Overheating |

The implementation of the cloud system facilitates the resolution of different categories of cloud failures. If these failures are not adequately addressed, there will be deterioration in availability, reliability, and quality of service, leading to financial losses for cloud users and cloud providers. This problem is mitigated by fault tolerance management, which enables the detection, identification, and handling of faults without compromising the result of CC [47]. Fault tolerance refers to the distributed system's capacity to continue operating smoothly and without interruption despite the failures in the component of the system [137]. Fault tolerance approaches can be classified into various categories such as fault aware policies (adaptive, hybrid, proactive, and reactive), redundancy techniques, and load balancing strategies. Figure 5.1 represents the various fault tolerance approaches.The adaptive techniques utilize predictive, learning,

and adaptive mechanisms to effectively handle and address new system failures.

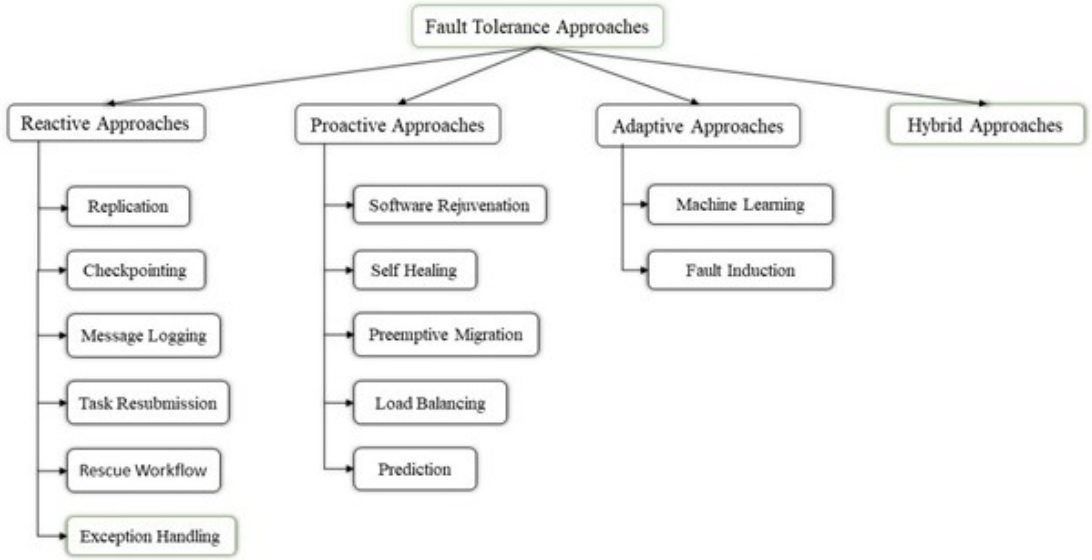



Figure 5.1: Fault Tolerance Approaches

Reactive techniques serve as a precautionary measure after the fault occurrence in the system. Proactive techniques aim to prevent or reduce the impact of system failure by making early predictions. The hybrid approaches integrate reactive, proactive, and adaptive techniques. This paper mainly focuses on proactive policies which are implemented through failure prediction models trained using ML and DL approaches. As there are various categories of cloud failures, this study emphasizes on prediction of job and task failures. Job and task failures are interconnected as a job can have one or more tasks.

This chapter focuses on two key experiments that investigate the application of DL, EL, HPT, and FL techniques for predicting task and job failures in CC environments. In the first part, a novel model is proposed that combines an ensemble of DL architectures with HPT to improve the prediction of both task and job failures in cloud data centers. In the second part, a FL framework is introduced for task failure prediction, ensuring scalability and privacy preservation across distributed environments. Both models were evaluated using the GCT v2011 dataset, providing insights into their effectiveness in enhancing failure prediction accuracy, reliability, and practical

applicability. This chapter is organized as follows. Section 2 presents the research methodology, including dataset description, data preprocessing steps (data cleaning, reduction, transformation, and imbalance handling), classification techniques, hyper-parameter tuning approaches, and performance evaluation measures adopted to assess model effectiveness. Section 3 describes the experimental framework and results of the optimized stacking ensemble of DL models for CFP (Study 1). Section 4 discusses the experimental framework and analysis of the FL based CFP model (Study 2). Finally, Section 5 provides the overall conclusion of the chapter.

## 5.2 Research Methodology

This section outlines the procedures and methodologies employed in the empirical study. It provides a structured overview of the steps followed to design, implement, and evaluate the predictive models developed in this research. Specifically, it discusses the dataset utilized, the preprocessing techniques used, classification techniques used, and performance evaluation measures used.

### 5.2.1 Dataset

In this study, we used Google cluster trace, released by Google in 2011, which consist of data of 29 days collected in month of May, 2011. The source data were retrieved from Google Cloud Storage (GCS), specifically GCT 2011 dataset ($\tilde{4}$00 GB). Access was automated using the urllib library, with compressed files extracted via gzip to generate intermediate job event and task event tables in CSV format. This dataset consists of information of 12,500 machines [2]. This dataset includes five files which are job events, machine events, task events, machine attribute, and task resource usage. This study focuses on job event and task event files. Job event file and task event file describes the lifecycle of the job or task. Each job is identified by job id but each task is identified by combination of job id and task index. Both files consist of data related to various events take place from job/task submission to job/task termination in event

type columns. Each job/task can go through 9 states from submission to termination. Both files also have one common field, scheduling class which shows how latency sensitive job/task is, whose value range from 0 to 3, where 0 represents non production job/task and 3 represents high latency sensitive job/task. Apart from this, task event has some more fields such as priority, requested resource data such as CPU, memory, and disk space. Each task is assigned a priority (0 represents the least priority task and 11 represents the highest priority task) in such an order that task with high priority gets more preference for resources rather than task having lower priority.

### 5.2.2 Data Preprocessing

There can be some noise or missing values in the dataset which may negatively effect the performance of the model. So, there is a need to perform preprocessing operation on the dataset before creating the model. Firstly, we clean the data by removing the missing values by using dropna function. For job failure data, the data of job event table is integrated with task event table into a single table. The most suitable values from the task event table is determined to combine with the record in the job event table. This is significant because a single job event can encompass several task events. To address this issue, data aggregation process is used where we combine sets of task event records for each job based on three specific features: CPU requests, memory requests, and disk space requirements. The objective is to determine the maximum value for each of them. The highest values are thereafter utilized to combine with the job event records. Task event failure data is generated with the help of task event table only. Next, we select the important features for job failure dataset and task failure dataset based on correlation matrix. Typically, features that have little to no correlation with the outcome of a job or task are eliminated. Figure 5.2 and figure 5.3 depicts the correlation matrix of the input features of task failure dataset and job failure dataset. Table 5.2 shows the independent and dependent features for task level and job level termination dataset. The independent features for task level termination dataset are CPU request, disk space request, memory request, priority, and scheduling class and

independent features for job level termination dataset are CPU request, disk space

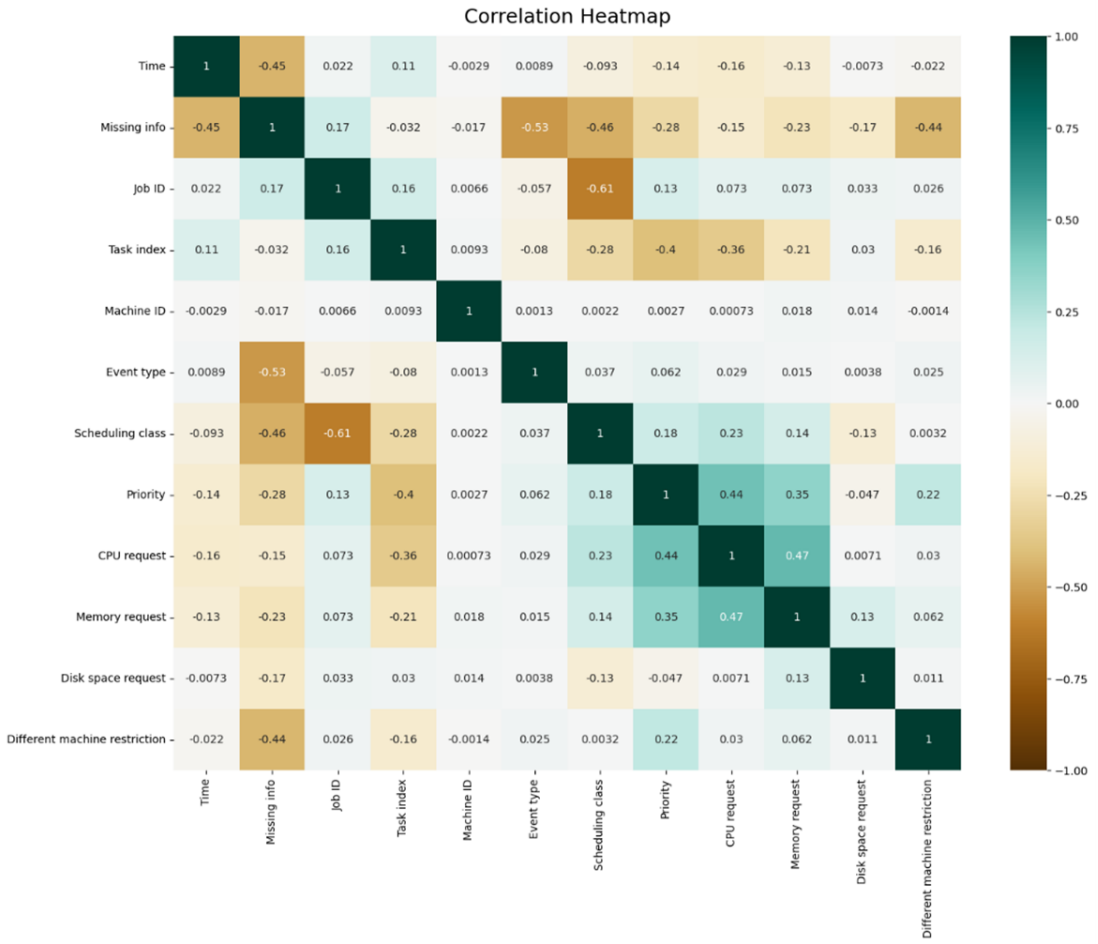



Figure 5.2: Correlation matrix of instances of dataset A

request, memory request, and scheduling class. Event type is the dependent feature for both job level and task level termination dataset. Event type column has 9 distinct values. We remove all those rows which have event type value 0, 1, 7, and 8 and convert the event type from numerical to categorical such that event type with value 4 is assigned 1(success) and other than 4 is assigned 0(failure). The task level and job level termination dataset have class imbalance problem. The ratio of failed instances to successful instances in the train dataset is calculated to identify imbalanced data, as the ratio of failed instances to the successful instances should ideally be near 1:1 to obtain balanced data [189]. To achieve balance in the data, SMOTE is employed in this study. The detailed explanation of SMOTE is given in Section 2.9 of Chapter 2. Through

these steps, two final datasets were obtained: Dataset A (job-level termination) and Dataset B (task-level termination), both cleaned, transformed, and balanced for robust model training.

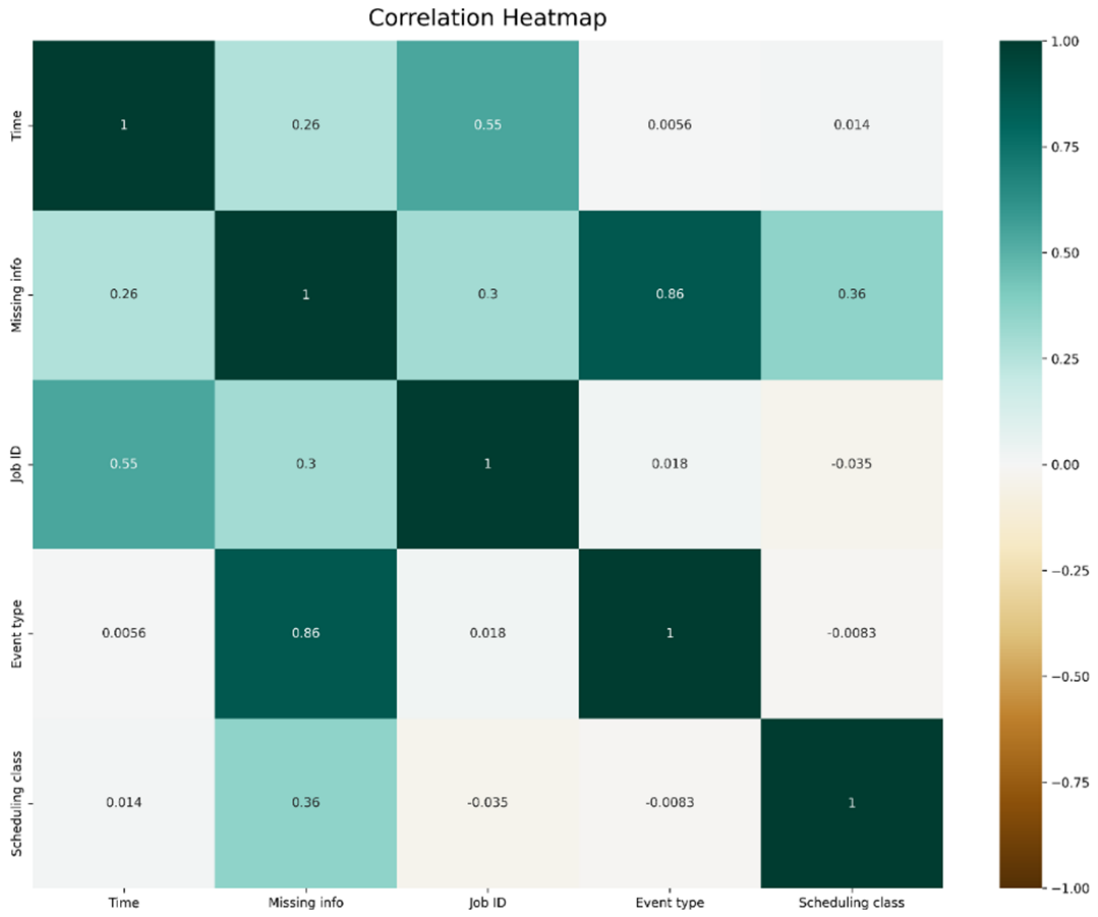



Figure 5.3: Correlation matrix of instances of dataset B

Table 5.2: Independent and dependent features for task level and job level termination dataset

| **Datasets** | **Independent Features** | **Dependent Feature** |
|---|---|---|
| **Task Level Termination Dataset** | CPU request, disk space request, memory request, priority, and scheduling class | Event Type (success/-failure) |
| **Job Level Termination Dataset** | CPU request, disk space request, memory request, and scheduling class | Event Type (success/-failure) |

### 5.2.3 Techniques

In this chapter, two main approaches are employed for CFP, each designed to improve the accuracy, scalability, and robustness of predictive models in large-scale cloud environments. Each study applies different classifiers to analyze and predict task and job failures in cloud data centers, based on distinct algorithmic techniques.

- Optimized Ensemble of DL Techniques

The first approach develops an ensemble framework (stacking) that combines multiple DL architectures such as CNN, LSTM, BiLSTM, and their hybrid variants (CNN-LSTM and CNN-BiLSTM) with RF as meta learner. The detailed description of DL models and stacking approach is given in Section 2.6 of Chapter 2. To further enhance predictive performance, HPT was applied using Optuna, an automated optimization framework.

Optuna is a HPT software framework designed to seamlessly implement a variety of cutting-edge optimization methods, enabling rapid and effective tuning of complex models. Developed by Preferred Networks, a Japanese AI company, Optuna automates the exploration of hyperparameter search spaces using a highly flexible define-by-run API, which allows users to construct dynamic and modular search spaces [13][121]. The framework supports a wide range of optimization algorithms, including grid search, random search, Bayesian optimization, and evolutionary algorithms, thereby offering versatility across different experimental needs. By default, Optuna employs the Tree-structured Parzen Estimator (TPE), a Bayesian optimization algorithm that balances exploration and exploitation efficiently. Additionally, Optuna incorporates pruning strategies, which terminate unpromising trials early to reduce computational cost and accelerate convergence. In this study, Optuna was applied to optimize critical hyperparameters such as dropout rate, number of epochs, batch size, and learning rate. Optuna was chosen because of its ability to efficiently navigate large and complex hyperparameter spaces, ensuring that the ensemble of DL models achieve near-optimal configurations. As a result, the tuned models demonstrated improved ac-

curacy, stability, and generalization, making Optuna a suitable choice for large-scale prediction tasks.

- ·Federated Learning

The second approach introduces a FL framework for predicting task and job failures in distributed cloud environments. FL enables collaborative training of models across multiple clients without requiring raw data to be centralized, thereby ensuring privacy preservation and scalability. Each client trains its local model on site-specific workload and failure data, and only the model updates are shared with a central server, where they are aggregated to form a global model. The detailed description is given in Section 2.6 of Chapter 2.

**Performance Evaluation Measure**

The predictive performance of the models is evaluated using accuracy, precision, recall, f1 score, AUC, and MCC. The detailed description and mathematical formula to compute these metrics is given in Section 2.11 of Chapter 2.

## 5.3 CFP with Optimized Ensemble of Deep Learning Architectures

This paper proposes a novel ensemble learning framework for job and task failure prediction in cloud environments. Our framework combines five complementary DL architectures, CNN, LSTM, BiLSTM, CNN-LSTM, and CNN-BiLSTM within a stacking ensemble, using a RF classifier as a meta-learner. To further enhance performance, we employ Optuna for automated HPT, covering parameters such as learning rate, dropout rate, batch size, and number of epochs. The proposed model is extensively evaluated on GCT v2011 dataset, with comparisons against both individual models and state-of-the-art baselines.

The main contributions of this study are as follows:

- An ensemble learning framework that integrates multiple DL models with a RF meta-learner for CFP.
- Systematic HPT using Optuna, ensuring robust and reproducible model performance.
- Comprehensive experimental evaluation, including comparisons with state-of-the-art baselines under consistent preprocessing.
- Computational efficiency evaluation in terms of training time, inference time, and memory usage.

Through this work, we demonstrate that integrating diverse learners with automated hyperparameter optimization significantly enhances CFP, paving the way for more reliable and fault-tolerant cloud services.

The following RQs are addressed in this study:

RQ1: Do proposed optimized ensemble model gives better performance as compared to single learning model in CFP problems?

RQ2: Does adding HPT techniques improves the performance of ensemble model and single learning model in CFP?

RQ1 examines the prediction performance of proposed EL model and base DL models. RQ2 is related to examining the prediction performance of proposed EL model and base DL model after adding the HPT approach.

### 5.3.1 Experimental Framework

Figure 5.4 shows the experimental framework used for implementation. The experimental pipeline begins by extracting and preprocessing job event and task event data, involving cleaning, aggregation, reduction, transformation, and class balancing. The resulting failure datasets are split into training and testing sets. Multiple DL architectures—including CNN, LSTM, BiLSTM, CNNLSTM, and CNNBiLSTM—are trained

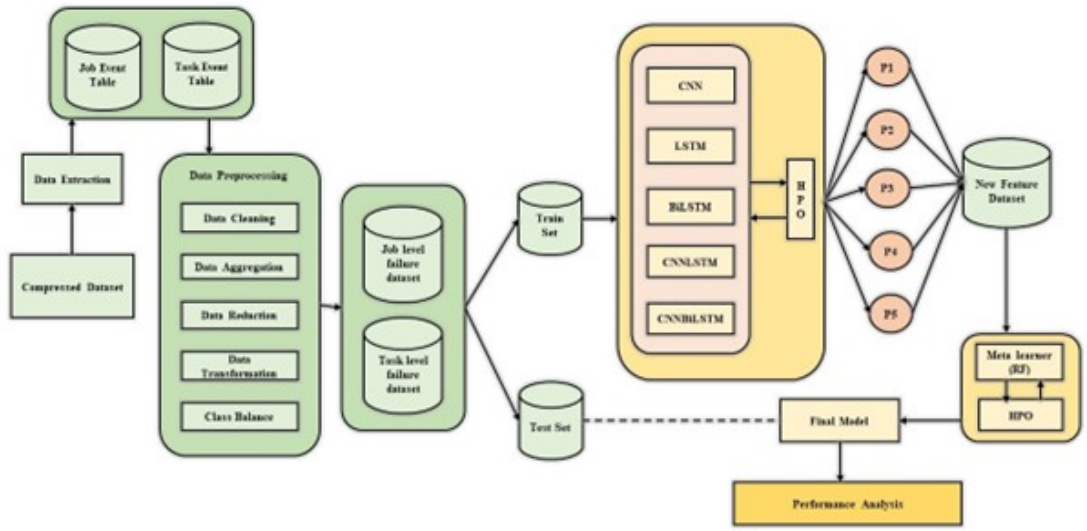



Figure 5.4: Proposed Framework

and optimized using HPO. Predicted outputs from these base learners are used to generate a new feature dataset, which serves as input to a meta-learner (RF) also tuned with HPT. The final ensemble model is evaluated on the test set, with comprehensive performance analysis carried out to assess the effectiveness of the approach.

### 5.3.2 Result Analysis

The experimental results based on the failure prediction for job level and task level termination dataset are described in this section. The preprocessed task level termination dataset samples and job level termination dataset samples were splitted into training and testing dataset for the purpose of training the aforementioned base models with HPT and testing the ensemble of those models respectively. Table 5.3 displays the experimental results of the base models and proposed model in terms of accuracy, AUC, precision, f1 score, recall and MCC for task level termination dataset. Table 5.4 displays the experimental results of the base models and proposed model in terms of accuracy, AUC, precision, f1 score, recall and MCC for job level termination dataset. Based on table 5.3 and table 5.4, the answer to RQ1 is presented.

- · *RQ1: Do proposed EL model gives better performance as compared to single*

*learning model in cloud failure prediction problems?*

According to table 5.3, we analyze that the proposed ensemble model outperforms base model and RF with an accuracy of 93.5%. AUC and MCC scores for proposed model are 0.943 and 0.806 respectively, which indicates that it can successfully predict the status of the task.

Table 5.3: Comparison between the performance of base models and proposed model for task level failure dataset

| **Model** | **Accuracy** | **Precision** | **Recall** | **F1 Score** | **AUC** | **MCC** |
|---|---|---|---|---|---|---|
| CNN | 90.8 | 0.732 | 0.985 | 0.839 | 0.934 | 0.794 |
| LSTM | 90.7 | 0.726 | 0.988 | 0.837 | 0.934 | 0.792 |
| BiLSTM | 90.9 | 0.733 | 0.984 | 0.840 | 0.935 | 0.795 |
| CNNLSTM | 90.7 | 0.731 | 0.984 | 0.839 | 0.934 | 0.794 |
| CNNBiLSTM | 89.96 | 0.719 | 0.959 | 0.822 | 0.919 | 0.769 |
| **Proposed Model** | **93.5** | 0.738 | 0.979 | **0.849** | **0.943** | **0.806** |
| Random Forest | 91.2 | **0.739** | **0.986** | 0.845 | 0.937 | 0.802 |

Furthermore, the F1 score (0.849) of the ensemble learning model is higher than those of the single learning models, suggesting that the ensemble learning models outperforms the single learning models in terms of precision and recall. RF has the second highest accuracy (91.2%), AUC (0.937), and MCC (0.802) scores. The results for proposed model and RF have a slight difference. The single learning model, including CNN, LSTM, BiLSTM, CNNLSTM, and CNNBiLSTM achieves lower prediction performance in terms of accuracy, AUC, precision, f1 score, recall, and MCC. Among the single learning models, BiLSTM achieves highest accuracy (90.9%), AUC (0.935), and MCC (0.795) scores.

According to table 5.4, we analyze that the proposed ensemble model outperforms base model and RF with an accuracy of 95%. AUC and MCC scores for proposed model are 0.965 and 0.898 respectively, which indicates that it can successfully predict the status of the task. Furthermore, the F1 score (0.849) of the ensemble learning model is higher than those of the single learning models, suggesting that the ensemble learning models outperforms the single learning models in terms of precision and recall. RF has the second highest accuracy (92.7%), AUC (0.935), and MCC (0.859) scores.

Table 5.4: Comparison between the performance of base models and proposed model for job level failure dataset

| Model | Accuracy | Precision | Recall | F1 Score | AUC | MCC |
|---|---|---|---|---|---|---|
| CNN | 92.3 | 0.935 | 0.932 | 0.933 | 0.922 | 0.843 |
| LSTM | 92.1 | 0.941 | 0.922 | 0.931 | 0.921 | 0.838 |
| BiLSTM | 90.9 | 0.908 | 0.939 | 0.924 | 0.904 | 0.814 |
| CNNLSTM | 91.2 | 0.921 | 0.929 | 0.925 | 0.909 | 0.819 |
| CNNBiLSTM | 90.1 | 0.933 | 0.894 | 0.913 | 0.903 | 0.799 |
| **Proposed Model** | **95** | **0.969** | **0.948** | **0.960** | **0.965** | **0.898** |
| Random Forest | 92.7 | 0.938 | 0.941 | 0.937 | 0.935 | 0.859 |

The single learning model, including CNN, LSTM, BiLSTM, CNNLSTM, and CNNBiLSTM achieves lower prediction performance in terms of accuracy, AUC, precision, f1 score, recall, and MCC. Among the single learning models, CNN achieves highest accuracy (92.3%), AUC (0.922), and MCC (0.843) scores.

Figures 5.5a, 5.5b, 5.5c, 5.5d, 5.5e, and 5.5f shows the comparison of results of base models, RF, and proposed model with or without HPT for task level termination dataset. Figures 5.6a, 5.6b, 5.6c, 5.6d, 5.6e, and 5.6f shows the comparison of results of base models, RF, and proposed model with or without HPT for job level termination dataset. Based on figure 5.5 and figure 5.6, answer to RQ2 is discussed.

- · *RQ2: Does adding HPT techniques improves the performance of EL model and single learning model in cloud failure prediction?*

Figure 5.5a shows the comparison of base model, RF, and ensemble models with or without HPT in terms of accuracy. Proposed model achieves 90.3% accuracy without HPT and after HPT, it achieves 91.6% accuracy. Figure 5.5b shows the comparison of base model, RF, and ensemble models with or without HPT in terms of precision. Proposed model achieves precision score of 0.807 precision without HPT and after HPT, it achieves 0.838 precision score. Figure 5.5c shows the comparison of base model, RF, and ensemble models with or without HPT in terms of recall. Proposed model achieves 0.897 recall score without HPT and after HPT, it achieves 0.979 recall score. Figure 5.5d shows the comparison of base model, RF, and ensemble models with

or without HPT in terms of f1 score. Proposed model achieves 0.849 f1 sore without HPT and after HPT, it achieves 0.859 f1 score. Figure 5.5e shows the comparison of base model, RF, and ensemble models with or without HPT in terms of AUC. Proposed model achieves AUC score of 0.909 without HPT and after HPT, it achieves 0.943 AUC score.

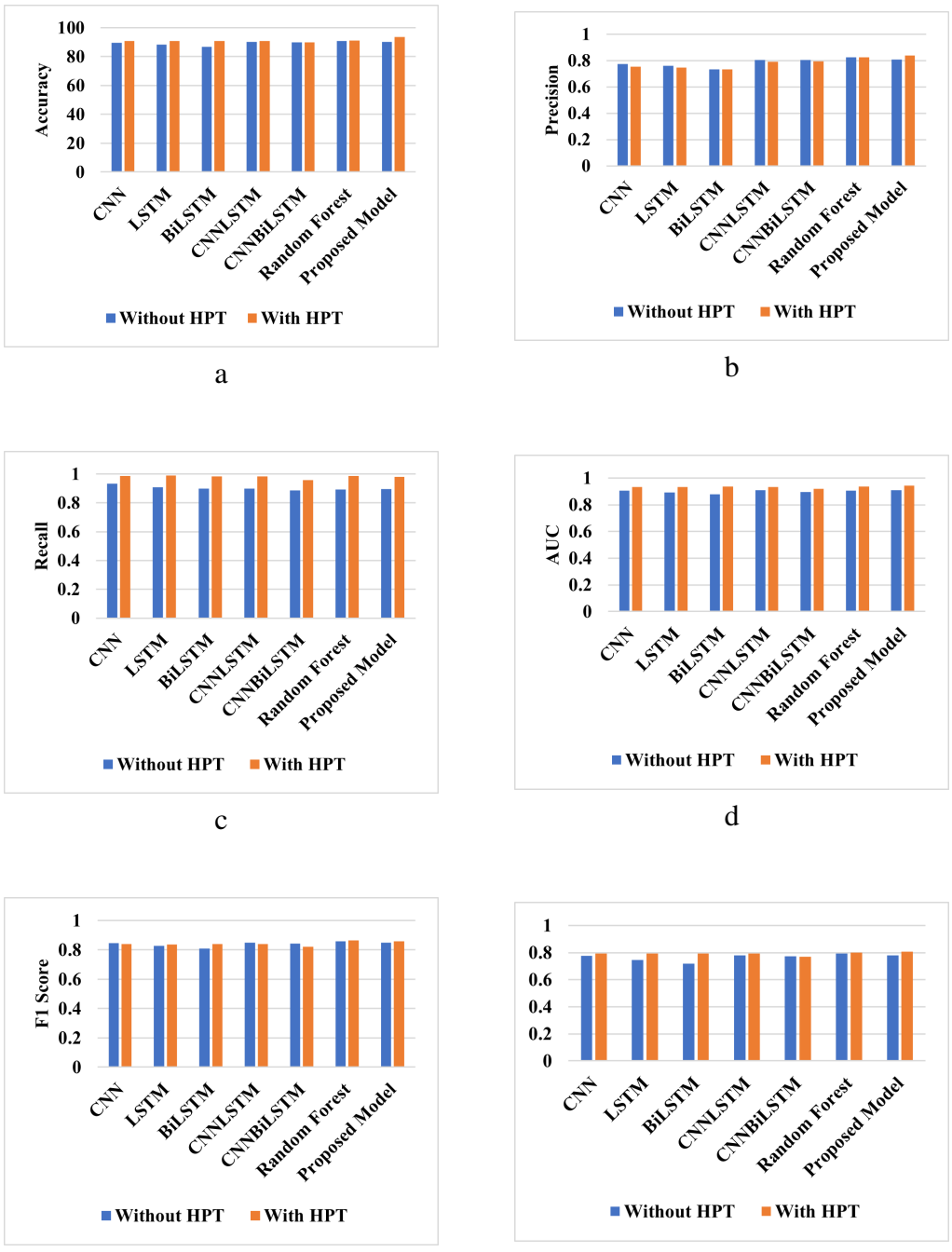
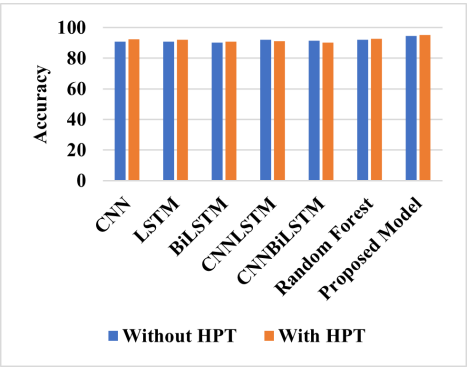


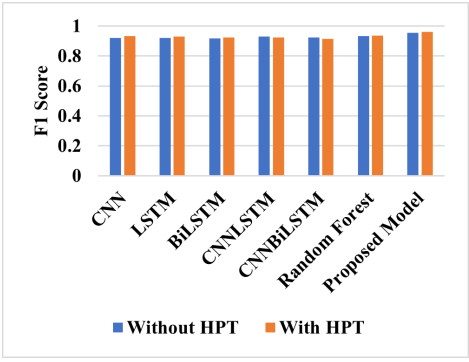

Figure 5.5: Comparison of results of base learning models, random forest, and proposed model with and without HPT for task level termination

Figure 5.5f shows the comparison of base model, RF, and ensemble models with or without HPT in terms of MCC. Proposed model achieves MCC score of 0.781 without HPT and after HPT, it achieves 0.806 MCC score. From the figures, we can analyze that there is a noticeable improvement across all single learning models, RF, and proposed model's performance metrics, highlighting the beneficial effects of HPT.

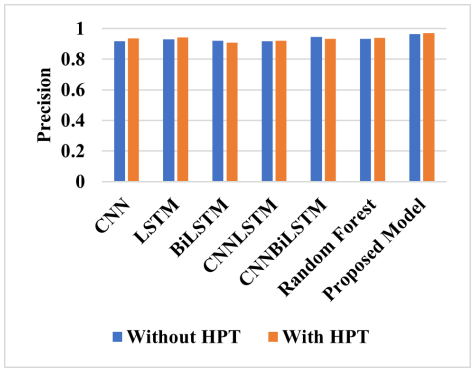
Figure 5.6a shows the comparison of base model, RF, and ensemble models with or without HPT in terms of accuracy. Proposed model achieves 94.5% accuracy without HPT and after HPT, it achieves 95% accuracy. Figure 5.6b shows the comparison of base model, RF, and ensemble models with or without HPT in terms of precision. Proposed model achieves precision score of 0.965 without HPT and after HPT, it achieves 0.969 precision score. Figure 5.6c shows the comparison of base model, RF, and ensemble models with or without HPT in terms of recall. Proposed model achieves 0.944 recall score without HPT and after HPT, it achieves 0.948 recall score. Figure 5.6d shows the comparison of base model, RF, and ensemble models with or without HPT in terms of f1 score. Proposed model achieves 0.956 f1 sore without HPT and after HPT, it achieves 0.96 f1 score. Figure 5.6e shows the comparison of base model, RF, and ensemble models with or without HPT in terms of AUC. Proposed model achieves AUC score of 0.95 without HPT and after HPT, it achieves 0.965 AUC score. Figure 5.6f shows the comparison of base model, RF, and ensemble models with or without HPT in terms of MCC. Proposed model achieves MCC score of 0.878 without HPT and after HPT, it achieves 0.898 MCC score. From the figures, we can analyze that there is a noticeable improvement across all single learning model, RF, and proposed model's performance metrics, highlighting the beneficial effects of HPT.



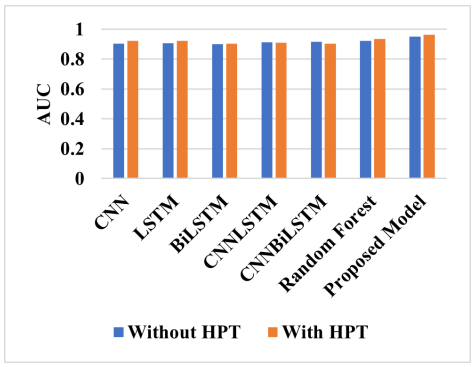

a



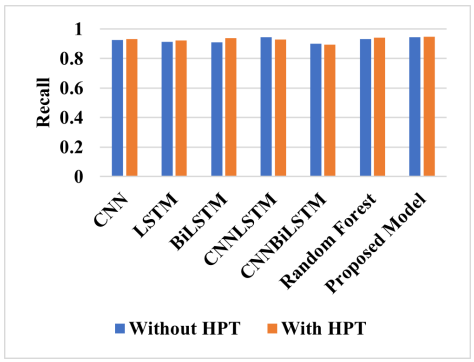

b



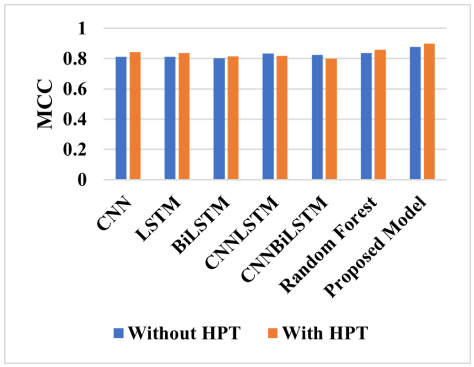

c




d




e




f

Figure 5.6: Comparison of results of base learning models, random forest, and proposed model with and without HPT for job level termination dataset

From these results, it is evident that HPT not only improves the ensemble model but also benefits the single learning models and RF. The improvements are more pronounced in the task-level dataset, where recall and AUC show significant gains, highlighting the importance of hyperparameter optimization in failure prediction tasks. All single models and RF also benefit from HPT, but the ensemble gains the most in recall and MCC, which are particularly important for failure detection. These findings confirm that HPT enhances both base and ensemble models, answering RQ2.

### Computational Efficiency

The computational efficiency of the proposed ensemble model was assessed in terms of training time, inference latency, and memory usage. As shown in figure 5.7, the ensemble requires higher resources than individual models, with training times of 115–120

minutes compared to 42–91 minutes for base learners, inference latency of 5.1–5.5 ms/sample versus 1.9–4.3 ms/sample, and memory usage of 7.4–7.9 GB compared to 3.1–6.0 GB. This overhead results from simultaneously training and maintaining five base learners before stacking. However, the increase in cost is modest relative to the substantial performance gains: the ensemble improves accuracy by up to 2.6% on the task-level dataset and 2.7% on the job-level dataset over the best single model, and by more than 2% over RF. Since training is typically performed offline in cloud environments, and inference latency of $\tilde{5}$ ms/sample remains feasible for real-time scheduling and failure recovery, the accuracy–efficiency trade-off strongly favors the ensemble model.

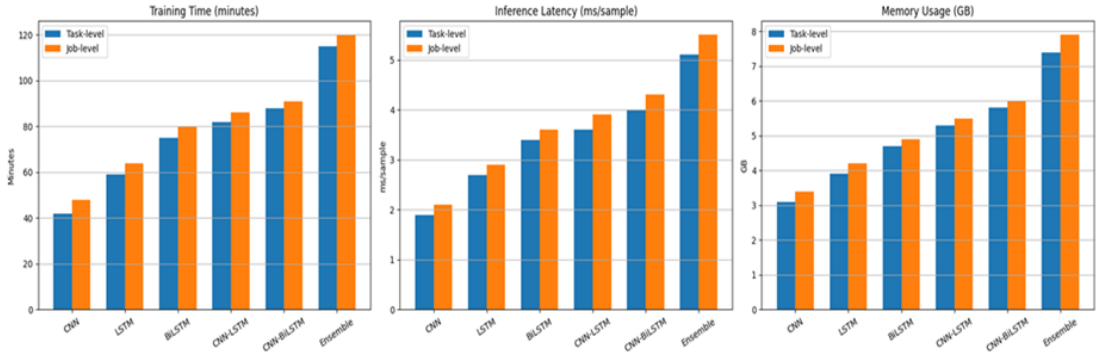



Figure 5.7: Computational Efficiency Comparison of Models (Task level vs Job Level)

### 5.3.3 Comparison with Existing Studies

To validate the effectiveness of the proposed model, we compared its performance against existing approaches reported in the literature. Table 5.5 summarizes the results of prior works along with the dataset partitioning strategies they employed.

Table 5.5: Comparative Analysis

| Study | Year | Technique | Accuracy | Description |
|---|---|---|---|---|
| Asmawi *et al.* [275] | 2022 | XGB | 94.35% | For job failure prediction (Dataset is splitted into 70% for model training and 30% for model testing) |
| | | RF | 89.75% | For task failure prediction (Dataset is splitted into 70% for model training and 30% for model testing) |

| | | | | |
|---|---|---|---|---|
| Gao *et al.* [77] | 2020 | BiLSTM | 87% | For job failure prediction (10-fold cross validation for data splitting) |
| | | | 93% | For task failure prediction (10-fold cross validation for data splitting) |
| Liu *et al.* [173] | 2017 | OS-ELM | 93% | For job failure prediction (Dataset is splitted into a training set and testing set in a ratio of 3:1) |
| Islam and Manivannan [111] | 2017 | LSTM | 81% | For job failure prediction (10-fold cross validation for data splitting) |
| | | | 87% | For task failure prediction (10-fold cross validation for data splitting) |
| Proposed Model | 2024 | EL model with HPT | 95% | For job failure prediction (Dataset is splitted into 80% for model training and 20% for model testing) |
| | | | 93.5% | For task failure prediction (Dataset is splitted into 80% for model training and 20% for model testing) |

From table 5.5, it is clear that the proposed ensemble model with HPT outperforms all compared studies:

- For job failure prediction, our model achieves 95% accuracy, surpassing the previous best method (XGB, 94.35% by Asmawi *et al.*).

- For task failure prediction, our model achieves 93.5% accuracy, exceeding BiLSTM (93% by Gao *et al.*), which was previously the strongest performer.

- Early methods such as OS-ELM (93% for job failures) and LSTM (81%–87%) are substantially weaker than the proposed approach.

The results confirm that the proposed ensemble learning approach with HPT consistently improves predictive performance across both job-level and task-level failure prediction. By integrating multiple DL classifiers and leveraging automated HPT, the model not only outperforms individual baselines but also surpasses prior state-of-the-art techniques in this domain.

## 5.4 CFP with Federated Learning Architecture

FL has emerged as a promising paradigm for building predictive models in distributed environments, particularly where data privacy and security are critical. In the context of CFP, FL enables multiple clients (e.g., cloud servers or data centers) to collaboratively train a model without sharing raw data, thereby safeguarding sensitive operational logs while still benefiting from collective learning.

In this study, we design and evaluate an FL-based architecture for predicting task failures in cloud data centers. The proposed architecture builds upon a central server model initialized with a CNN. The training process begins with the server distributing the global model to multiple clients. Each client further trains the model on its local private dataset and updates its parameters independently. These locally trained parameters are then aggregated on the central central server using the FedAvg algorithm to update the global model. This iterative process continues until the model converges to a stable and high-performing predictor.

This architecture ensures privacy by keeping data local, enhances scalability through collaborative training across multiple clients, and achieves high predictive performance, with the FL-CNN model outperforming standalone CNN in terms of accuracy, precision, recall, f1 score, AUC, and MCC. Thus, FL provides an effective framework for building robust and secure predictive models for large-scale cloud data centers.

### 5.4.1 Experimental Framework

This study deals with only task level termination dataset. For the task-level termination dataset, failure data is generated directly from the task event table. The independent features considered are CPU request, disk space request, memory request, priority, and scheduling class, while the event type serves as the dependent feature. The detailed description of dataset is given in 5.2.1 . The detailed description of data preprocessing(reduction, transformation, cleaning, and imbalance handling) is given in 5.2.2. Figure 5.8 shows the methodology used to perform the experimentation.

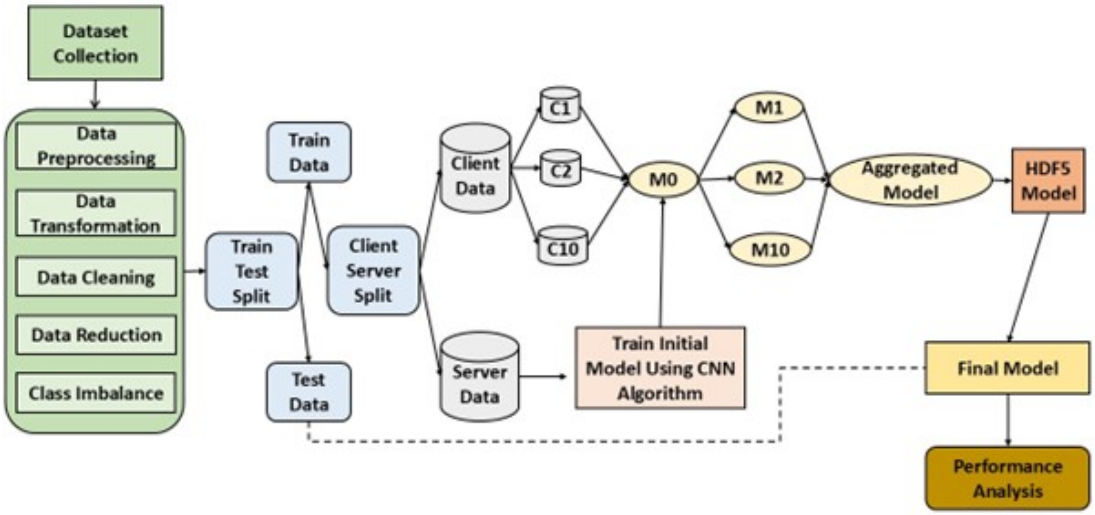



Figure 5.8: Proposed Methodology

### 5.4.2 Result Analysis

The experimental results based on the failure prediction for task level termination dataset are discussed. The preprocessed task level termination dataset samples were splitted into training and testing dataset for the purpose of training the CNN model and FL model consisting CNN as base model and testing the performance of the model on unseen data respectively. Table 5.6 presents the experimental results of both models in terms of accuracy, AUC, precision, f1 score, recall and MCC for task level termination dataset. We analyze that the proposed FL model consisting CNN as base model outperforms CNN model with an accuracy of 93.9%. AUC and MCC scores for proposed model are 0.956 and 0.836 respectively, which demonstrates that it can successfully predict the status of the task. Additionally, the F1 score (0.954) of the proposed model is higher than CNN model, suggesting that the proposed models outperform CNN model in terms of precision and recall.

Table 5.6: Performance comparison between CNN and Federated Learning with CNN.

| Model | Accuracy | Precision | Recall | F1 Score | AUC | MCC |
|---|---|---|---|---|---|---|
| CNN | 90.3% | 0.925 | 0.903 | 0.908 | 0.928 | 0.783 |
| FL with CNN | 93.9% | 0.962 | 0.929 | 0.954 | 0.956 | 0.836 |

## 5.5 Conclusion

This chapter presented two complementary approaches for CFP, each addressing different aspects of predictive modeling and deployment. The first study demonstrated that an optimized stacking ensemble of DL models, enhanced through HPT, significantly improves prediction accuracy and robustness compared to individual models. By effectively combining CNN, LSTM, and BiLSTM architectures, the ensemble captured both spatial and temporal dependencies in workload data, yielding superior performance across multiple evaluation metrics. The second study extended this work by introducing a FL framework for CFP, designed to overcome the limitations of centralized training. Using CNN and ANN as base models, the FL approach preserved data privacy by ensuring that raw workload and failure data remained distributed across clients. Experimental results confirmed that FL-based models achieved predictive performance comparable to centralized approaches, while also offering scalability and privacy benefits essential for real-world deployment. Together, these studies establish that both ensemble learning and FL provide powerful pathways for advancing CFP. While ensemble models maximize predictive performance through architectural diversity and optimization, federated frameworks ensure privacy-preserving, scalable, and secure deployment in distributed cloud environments. These insights highlight the dual importance of accuracy and practicality, paving the way for more reliable and trustworthy CC infrastructures. Looking ahead, the next chapter builds on these foundations by extending the analysis to SDP models, with a particular focus on their security and performance evaluation in deployment environments, thereby broadening the scope of predictive modeling in software engineering.

# Chapter 6

# SECURITY AND PERFORMANCE ANALYSIS OF DOCKERIZED SOFTWARE DEFECT PREDICTION MODEL

## 6.1 Introduction

The Software Development Lifecycle (SDLC) consists of software testing that is time and resource consuming. The testing procedure aims to deliver software that is completely error-free and meets the needs of all stakeholders. Testing adds to the overall project budget because it is an expensive process. The effectiveness and quality of software improve when defects are correctly predicted in an early stage [9]. Finding software defects is a necessary but expensive process. The cost of SDLC increases significantly due to software defects, as later defect detection leads to higher correction costs. Early identification of defects through SDP helps minimize testing and maintenance expenses by focusing efforts on defect-prone components. Various ML, DL, and ensemble learning techniques have been explored to improve SDP accuracy, mak-

ing software systems more reliable and cost-effective. However, ensuring data privacy and secure collaboration remains a major challenge when defect data originate from multiple organizations.

FL offers a practical and privacy-preserving solution for SDP in real-world scenarios where sharing raw repository data is constrained by confidentiality, licensing, or regulatory restrictions. In industrial and cross-organization collaborations, partners may be unable or unwilling to expose proprietary code metrics or process data due to intellectual property concerns, non-disclosure agreements, and privacy regulations. FL circumvents these barriers by enabling participants to collaboratively train predictive models through the sharing of model parameters or updates, rather than sensitive raw data, thereby safeguarding proprietary assets and streamlining compliance with legal frameworks. Moreover, FL supports incremental participation by geographically distributed teams, facilitating bug prediction workflows that respect organizational boundaries and local data controls. These collaborative attributes of FL improve trust and broaden the pool of available code and defect metrics, which are essential for robust defect prediction but otherwise siloed. Additionally, decentralizing model training in FL reduces risks and legal hurdles typically associated with centralizing sensitive software assets, making defect prediction feasible for partners bound by strict privacy or data governance requirements.

Recognizing the opportunities and deployment challenges, this study investigates FL's feasibility for SDP, focusing not only on model accuracy but also on the security of deployed FL solutions within containerized environments, which is essential for safe, scalable integration in organizational ecosystems. The models are containerized using Docker to ensure scalability, reproducibility, and ease of deployment across heterogeneous environments. In addition to predictive performance, this study also evaluates resource utilization, analyzing CPU, memory, network, and disk usage during model execution to ensure computational efficiency. Furthermore, a comprehensive security vulnerability assessment of the Dockerized FL models is conducted using open-source tools such as Anchore, Aqua Trivy, Clair, and JFrog Xray. Overall, this chapter presents a secure, resource-efficient, and scalable FL-based SDP framework suitable

for real-world software engineering applications.

This chapter presents four key experiments that collectively explore the integration of metaheuristic optimization, HPT, and FL techniques for SDP. It also emphasizes the deployment of predictive models through Docker containerization and evaluates their performance in terms of security and resource efficiency using open-source analysis tools. The first part focuses on assessing security vulnerabilities in official and verified Docker images using multiple container scanning tools. The second part introduces a CNN-based SDP model optimized through a hybrid metaheuristic-driven HPT approach to enhance predictive accuracy. The third part investigates the application of FL in developing decentralized SDP models, which are then containerized using Docker for detailed analysis of security vulnerabilities and resource consumption. The final part proposes an optimized FL-GWO model for SDP, combining FL with the GWO and further evaluates its containerized deployment through security vulnerability analysis.

This chapter is organized as follows: Section 2 presents the research methodology, including dataset description, data preprocessing steps (data cleaning, reduction, transformation, and imbalance handling), classification techniques, HPT approaches, performance evaluation measures adopted to assess model effectiveness, tools used for security vulnerability analysis, and tools used for resource consumption analysis. Section 3 describes the experimental framework and result of security analysis of official and verified docker images (study 1). Section 4 describes the experimental framework and result of SDP with optimization of DL techniques using hybrid metaheuristic approach (study 2). Section 5 describes security and performance analysis of dockerized FL model for SDP (study 3). Section 6 describes security analysis of optimized FL model for SDP (study 4). Finally, section 7 describes the conclusion of the chapter.

## 6.2 Research Methodology

This section outlines the procedures and methodologies employed in the empirical study. It provides a structured overview of the steps followed to design, implement,

and evaluate the predictive models developed in this research. Specifically, it discusses the dataset utilized, the preprocessing techniques used, classification techniques used, HPT used, performance evaluation measures used, tools used for security analysis, and tools used to evaluate resource consumption by the model in terms of CPU utilization, memory utilization, disk i/o utilization, and network bandwidth utilized.

### 6.2.1 Dataset

The dataset is collected from the publicly available repository [184][261] which consists of both static code metrics and process metrics. We choose datasets developed using java language since they contain static code metrics such as WMC, DIT, LOC, LCOM, etc., and process metrics such as NR, NDC, NML, and NDPV [184]. A detailed description of the static code metrics and process metrics is given in 2.2. Table 6.1 lists the datasets that were used in this study.

Table 6.1: Statistical data about datasets

| **Dataset** | **# of features** | **# of instances** | **# of buggy instances** | **# of non buggy instances** | **Bug ratio** |
|---|---|---|---|---|---|
| Ant1.6 | 25 | 524 | 92 | 432 | 17.56% |
| Ant1.7 | 25 | 1065 | 166 | 899 | 15.59% |
| jEdit4.1 | 25 | 644 | 79 | 565 | 12.27% |
| Xalan2.5.0 | 25 | 945 | 387 | 558 | 40.95% |
| Xalan2.6.0 | 25 | 1170 | 411 | 759 | 35.13% |
| Xerces1.2.0 | 25 | 515 | 71 | 444 | 13.79% |
| Xerces1.3.0 | 25 | 545 | 69 | 476 | 12.66% |
| Xerces1.4.4 | 25 | 671 | 437 | 234 | 65.13% |

### 6.2.2 Data Preprocessing

Datasets obtained from publicly available repositories may contain missing values that can negatively impact the performance of the created model. Hence, preprocessing is done on datasets to avoid this type of problem. This can include eliminating data points with missing values if they are extremely small in count, or replacing missing

values with mean value or median value or with some user defined constant. There is an imbalance class problem in the datasets chosen. The imbalanced data identification is done by taking the ratio of a number of Non-Defective and Defective Classes in the train dataset since the ratio should be nearly equal to 1: 1 for balanced data. So, random oversampling of the data is done to balance it. In random oversampling, the weightage of the class which is low in the count is increased by repetition of that class's data points randomly. The defective class is very low in count in all the datasets used, so it should be given more weightage by repetition to balance the dataset. If the ratio of non-defective to defective class is X: 1 then defective class's data point will be repeated X times to balance the data and to avoid biasness of the majority class. Figure 6.1 shows the data before oversampling and after oversampling. Random Over Sampling of the minority class is done (only during the training of the model).

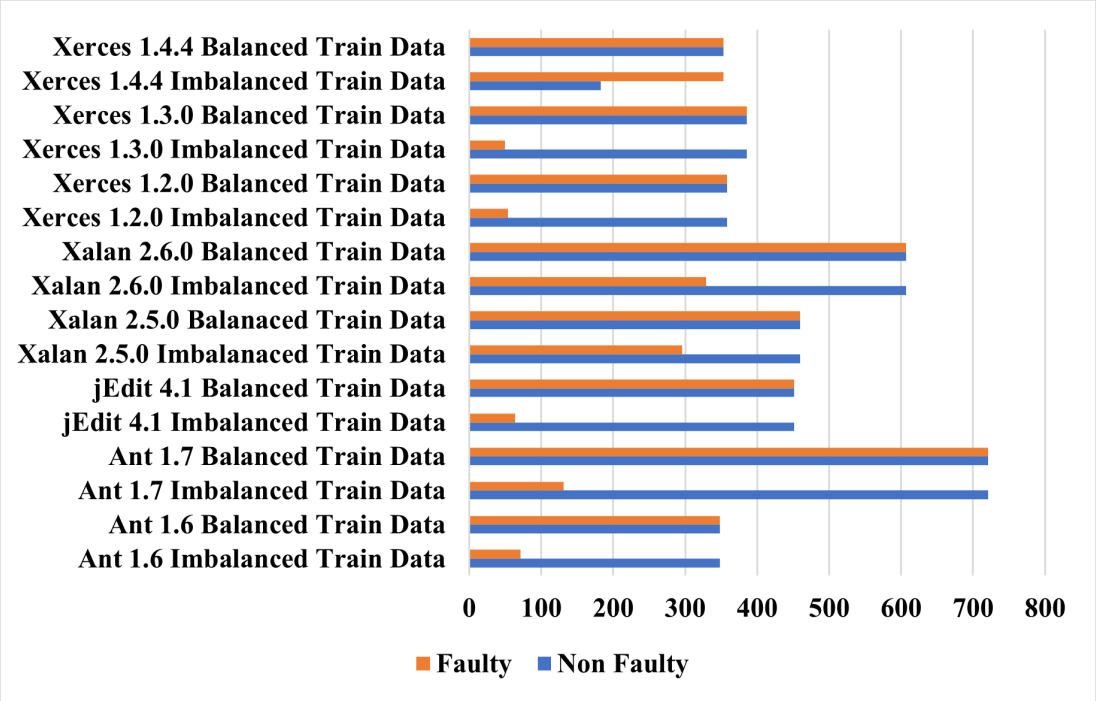



Figure 6.1: Before oversampling and after oversampling

### 6.2.3 Techniques

In this chapter, three main approaches are employed for SDP, each designed to improve the accuracy, scalability, and robustness of predictive models. Each study applies dif-

ferent classifiers to analyze and predict software defects based on distinct algorithmic techniques.

- SDP with Optimization of DL Techniques using Hybrid Metaheuristic Approach

In this study, the hyperparameters of CNN is optimized using hybrid of GWO and WOA. CNN is used for predicting the software defects and GWO_WOA is used to optimize the hyperparameters of CNN algorithm. CNN is a deep ANN that consists of many layers of neural networks. Each neural network layer consists of neurons. A CNN model is constructed using several layers, including the convolutional, batch normalization, pooling, dropout, and dense layers. The architecture view of the CNN model used in this model comprises of three levels of convolution each with two 2DCNN layers and filters of different kernel sizes in each layer, max pooling layer with a pool size of two, dropout layer to minimize the likelihood of overfitting. This study employs the early stopping technique to halt the model when training outcomes are stagnant, such as when continuing training produces minimal improvement or when continuous training rounds produce no change, to avoid overfitting.

In this study, a hybrid meta-heuristic optimization technique, which is a combination of GWO and WOA, is applied to optimize the hyperparameters of CNN for SDP. It comprises the best characteristic of both GWO and WOA. WOA struggles with performance during the exploitation phase and often becomes trapped in local optima [17][208]. In contrast, GWO has shown strong performance in the exploitation phase, particularly when tested on unimodal benchmark functions, making it highly competitive among common metaheuristic algorithms [209][258]. Thus, this paper aims to hybridize GWO with WOA to address these limitations, leveraging GWO's strengths in effectively exploiting optimal solutions. The hyperparameters tuned are dropout rate, epoch count, learning rate, and batch size. Table 6.2 shows the details of the hyperparameters values of the proposed model for SDP.

In this study, FL approach for predicting software defects is used. We use DL models CNN and ANN as base models. Firstly, base DL model CNN/ANN is trained using server data to build the global model (M0). The global model is sent to each client

and clients train the global model using their local data on their device to build a local model (M1, M2. . . , M10).

Table 6.2: Hyperparameter Values

| **Component** | **Hyperparameter Values** |
|---|---|
| Activation Functions | ReLU, Sigmoid (At last level) |
| Dropout rate | 0.2–0.5 |
| Epoch count | 5, 10, 20, 50 |
| Learning rate | 0.001–0.01 |
| Batch size | 8–32 |
| Optimizers | Adam |
| Verbose | 1 |
| Number of layers | 6 2DCNN + 2 Dense Layer + 1 Output Layer |
| Loss function | Binary cross entropy |

- Security and Performance Analysis of Dockerized FL Model for SDP

All the local model weights are aggregated using FedAvg method to build a global model and then the global model is sent back to the server. The performance of ANN based FL model and CNN based FL model is compared. The detailed description of ANN, CNN, and FL is given in section of chapter 2. We use Adam optimizer with relu activation function for all the layers except the output layer. As the dependent variable has binary classes 0 or 1, so sigmoid activation function is used in the output layer. Table 6.3 shows hyperparameter details of the proposed framework for predicting software defects.

Table 6.3: Hyperparameter details

| **Components of the Proposed Framework** | **Hyperparameters** |
|---|---|
| Number of iterations | 20 |
| Number of clients | 10 |
| Verbose | 0 |
| Epochs | 100 |
| Batch size | 10 |
| Activation Function | ReLU, Sigmoid (At last level) |
| Optimizer | Adam |
| Loss function | Binary cross entropy |
| Number of Convolution 2D layers | 3 |
| Regularization Method | Dropout |

- Security Analysis of Optimized FL Model for SDP

In this study, we propose two FL models enhanced with HPT: FLANN with HPT and FLCNN with HPT. The hyperparameters of ANN and CNN algorithms are optimized using GWO. The detailed description of FL, ANN and CNN is given in section of chapter 2. ANN and CNN algorithms are used for predicting the software defects and GWO is used to tune the hyperparameters of ANN and CNN algorithms.

Inspired by grey wolves, Mirjalili *et al.* [209] proposed GWO, a meta heuristic optimization system, in 2014. The algorithm simulates the leadership hierarchy of grey wolves in a pack. The hierarchy consists of four types of wolves:

- Alpha Wolves: The best solutions (leaders) that guide the pack.
- Beta Wolves: The second-best solutions that support the alphas and help in decision-making.
- Delta Wolves: The third-best solutions that follow the alphas and betas and assist in hunting.
- Omega Wolves: The weakest solutions that follow the pack and do not influence decisions.

The GWO algorithm operates effectively because of three fundamental motivating factors: finding, approaching, and hunting the target [294]. GWO has been applied in various fields such as economic dispatch problem [124], power system [218], gene classification, clustering, and selection [64], prediction problems [215], parameter tuning [135], etc. The hyperparameters tuned are dropout rate, epoch count, learning rate, and batch size. Table 6.4 shows the details of the hyperparameters values of the proposed model for SDP.

Table 6.4: Hyperparameter values of the proposed model

| **Hyperparameter** | **Value** |
|---|---|
| Global epoch count | 20 |
| Number of clients | 10 |
| Activation functions | ReLU, Sigmoid (At last level) |
| Loss function | Binary cross entropy |
| Dropout rate | 0.05–0.5 |
| Local epoch count | 5, 10, 20, 50 |
| Learning rate | 0.001–0.01 |
| Batch size | 32–256 |
| Optimizers | Adam |

### 6.2.4 Performance Evaluation Measure

The predictive performance of the models is evaluated using accuracy, precision, recall, f1 score, AUC, and MCC. The detailed description and mathematical formula to compute these metrics is given in Section of Chapter 2.

### 6.2.5 Tools for Security Vulnerability Analysis

There can be some security issues in federated learning models as some packages can be outdated or may have some vulnerabilities. To select container scanning tools, we followed a set of criteria. We choose widely-used, freely-available tools that have been utilized by practitioners or researchers. Free tools are considered due to the limited capabilities of trial versions of paid products. Here, Anchore, Aqua Trivy, JFrog Xray, Docker Scout, and Docker scan are used to detect security vulnerabilities in the docker image which consists of all the files, libraries, and packages needed in building the federated learning model for software defect prediction. We have installed extensions of tools used in this study on the docker desktop and then the created docker image is scanned to detect security vulnerabilities. Table 6.5 lists the features of container scanning tools used. A detailed description of anchore, aqua trivy, jfrog xray, docker scan, and docker scout is given in section 2.14 of chapter 2.

Table 6.5: Container scanning tools used

| Tool Name | OS Packages | Non OS Packages | Database Information |
|---|---|---|---|
| Anchore | Yes | Yes | No |
| Aqua Trivy | Yes | Yes | Yes |
| JFrog Xray | Yes | Yes | Yes |
| Docker Scan | Yes | Yes | No |
| Docker Scout | Yes | Yes | Yes |

### 6.2.6 Tools for Resource Consumption Analysis

To select container performance monitoring tools, we followed a set of criteria. We choose widely-used, freely-available tools that have been utilized by practitioners or researchers. Free tools are considered due to the limited capabilities of trial versions of paid products. Container performance monitoring tools such as docker stats, cAdvisor, Prometheus, Grafana, sematext, and locust are used in this study. Prometheus is configured to scrape relevant metrics exposed by docker and the application itself. cAdvisor is deployed as a container runtime monitor, collecting low-level resource utilization data. Grafana is used for visualizations and dashboarding purposes, providing real-time performance insights. The monitoring data provides valuable insights into the performance characteristics and resource demands of the containerized environment throughout our experiments.

1. Docker Stats: It is a built-in command of docker to compute resource usage of docker containers. When we execute the ‘docker stats’ command it returns the container ID, container name, CPU utilization, memory utilization, network i/o utilization, and disk i/o usage of all the running containers on docker.

2. cAdvisor: cAdvisor stands for Container Advisor. It was first introduced by Google and now it is a part of CNCF (Cloud Native Computing Foundation). This tool supports real-time monitoring of running containers and provides a web-based interface to view performance metrics with the help of visualizations and graphs. It can work on multiple platforms such as docker and kubernetes.

3. Prometheus: It was first introduced by Sound Cloud and now it is a part of CNCF

(Cloud Native Computing Foundation). It is a monitoring and alerting tool used to monitor the health and performance of the containers. It stores the collected data in a time series database which enables users to view and analyze historical data over time. This tool also provides query language known as PromQL that allow users to query and analyze the collected data using various function and operators.

4. Grafana: It was first introduced by Torkel Ödegaard in 2014 and now it comes under Grafana labs. It supports various data sources such as Prometheus, influxdb, and elastic search to create real-time dashboards and visualizations of data. It provides a web-based interface to create and share visualizations and it supports a wide range of visualization types such as graphs, charts, tables, and maps. Grafana dashboards were designed to display key metrics and performance indicators, allowing us to monitor the system's behavior and identify potential performance bottlenecks.

5. Locust: It is an open-source load-testing tool used for testing and optimizing the performance of APIs and web applications. This tool can be integrated with other tools such as Prometheus and Grafana for monitoring and visualizing the results of load tests. The load test can be executed with multiple virtual users, which simulates real-world traffic and stress tests the web applications.

6. Sematext: It is an open-source log monitoring tool that runs as a container and collects resource usage of all the containers. It automatically identifies containers running on docker and then starts the container monitoring process. It enables us to monitor resource usage metrics, logs, and top containers in a single dashboard.

## 6.3 Security Analysis of Official and verified docker Images

Docker is a widely used containerization platform that streamlines software deployment by packaging applications and dependencies into portable containers. Docker hub, its central repository, hosts both official and community images, allowing developers to share and reuse images easily. However, since docker images are maintained by their respective users, Docker itself has limited control over update frequency and maintenance quality. Many images—particularly verified images—may remain outdated for extended periods, leaving them vulnerable to security risks due to obsolete libraries or unpatched dependencies. Such vulnerabilities can compromise container integrity and expose systems to potential exploits.

The objective of this study is to systematically analyze and compare security vulnerabilities in official and community docker hub images using multiple open-source tools. This investigation aims to identify differences in vulnerability prevalence and severity across image types, providing insights into the overall security posture of containerized software environments.

The following RQs are addressed in this study:

RQ1: Is there any clear difference between the number of vulnerabilities in official and community docker images?

RQ2: Is there any difference between the type of vulnerabilities found in official and community docker images?

RQ3: How many CVEs were published per year in NVD database?

This study aims to analyze the security landscape of Docker images by combining three key research objectives. RQ1 and RQ2 investigates whether there is a significant difference in the number and types of vulnerabilities between official and community Docker images, providing insights into their relative security, maintenance quality, and potential risks. Furthermore, RQ3 examines the trend of CVEs published annually

in the NVD to understand how security threats related to containerized environments have evolved over time. Together, these analyses offer a comprehensive view of the vulnerability patterns and security posture of Docker images in real-world deployment scenarios.

### 6.3.1 Experimental Framework

This section describes the steps followed to detect the vulnerabilities in docker images, tools used and images selected for analysis. This study follows an explorative methodology with a quantitative approach to collect and analyze the data. We use container scanning tools to detect the security vulnerabilities in official and community docker images. We collect number of vulnerabilities identified in docker images, severity of identified vulnerabilities, unique CVEs, and the year in which CVE is published using container scanning tools. After collecting this data, we analyze this data to find the differences in container scanning tools, the differences in vulnerabilities in official and community docker images, and how many CVEs get published per year

**Selection of Container Scanning Tools**

We follow certain criteria to select the container scanning tools. In this study, we chose those tools that are popular, free of charge, and have been used in earlier studies. Since trial versions of paid tools have limited functionality therefore, we consider free of charge tools. We use Anchore, Aqua Trivy, Docker scan, and Jfrog Xray to identify the vulnerabilities in docker images.

**Selection of Docker images**

Docker hub consists of a mixture of official and verified images. We consider both official and verified images on docker hub. Official images are published by docker team itself and verified images are published by a certified publisher. In this study, we consider top 5 official images and top 5 verified images for each selected official image from docker hub. Images are selected based on number of pulls, updation time period and number of stars. We choose ubuntu, nginx, mongo, mysql, and postgres official

docker images and their corresponding top 5 verified images from docker hub. Table 6.6 shows selected official images, their number of downloads, number of stars, and latest updation time.

Table 6.6: Official Images

| Name | Number of Pulls | Number of Stars | Updated |
|---|---|---|---|
| Mongo | 1B+ | 9.4K | 12 hours ago |
| Nginx | 1B+ | 10K+ | 17 hours ago |
| Ubuntu | 1B+ | 10K+ | 14 hours ago |
| Mysql | 1B+ | 10K+ | A day ago |
| Postgres | 1B+ | 10K+ | A day ago |

We use the following steps to detect the vulnerabilities in docker images:

- Download and install docker
- Download the selected images using docker pull command.
- Download and install the selected tools
- Start the scan of the docker image
- Save the results such as image name, tool name, total number of vulnerabilities identified, and severity wise vulnerabilities.
- Extract the CVE publication year in order to find number of CVEs published per year.

### 6.3.2 Result Analysis

In this section, we describe the results collected after each scan of the images using container scanning tools. Table 6.7 and fig 6.2 shows the total number of vulnerabilities reported by each tool in selected official and verified images. According to table 2, there is a major difference between the total number of vulnerabilities detected by each tool. Although Trivy identified the highest total number of vulnerabilities (2300), anchore identified 2227 vulnerabilities, jfrog xray identified 1332 vulnerabilities, and

docker scan identified the lowest total number of vulnerabilities (1213) in selected docker official and verified images (RQ1).

Table 6.7: Total number of vulnerabilities reported by each tool

| | **Anchore** | **Trivy** | **Docker Scan** | **JFrog Xray** |
|---|---|---|---|---|
| Postgres | 112 | 116 | 52 | 50 |
| bitnami/postgresql | 101 | 104 | 52 | 32 |
| circleci/postgres | 208 | 211 | 45 | 17 |
| bitnami/postgres-exporter | 81 | 86 | 45 | 18 |
| rapidfort/postgresql | 34 | 34 | 12 | 4 |
| bitnami/postgresql-repmgr | 101 | 104 | 52 | 32 |
| Ubuntu | 18 | 15 | 9 | 1 |
| ubuntu/apache2 | 0 | 12 | 9 | 1 |
| ubuntu/squid | 0 | 6 | 4 | 0 |
| ubuntu/cortex | 77 | 63 | 14 | 5 |
| ubuntu/redis | 20 | 14 | 8 | 3 |
| ubuntu/prometheus | 74 | 82 | 33 | 12 |
| mongo | 82 | 58 | 32 | 10 |
| bitnami/mongodb | 80 | 58 | 33 | 9 |
| circleci/mongo | 92 | 216 | 85 | 17 |
| bitnami/mongodb-exporter | 83 | 53 | 21 | 15 |
| rapidfort/mongodb | 42 | 35 | 18 | 25 |
| percona/percona-server-mongodb | 20 | 24 | 17 | 19 |
| Mysql | 70 | 67 | 33 | 13 |
| circleci/mysql | 231 | 216 | 83 | 17 |
| bitnami/mysql | 80 | 81 | 43 | 11 |
| bitnami/mysqld-exporter | 83 | 73 | 36 | 12 |
| rapidfort/mysql-ib | 21 | 21 | 12 | 3 |
| rapidfort/mysql | 11 | 11 | 8 | 4 |
| Nginx | 140 | 144 | 102 | 23 |
| bitnami/nginx | 80 | 86 | 53 | 16 |
| bitnami/nginx-ingress-controller | 93 | 96 | 56 | 17 |
| kasvubey/nginx | 79 | 79 | 52 | 16 |
| rapidfort/nginx | 35 | 34 | 27 | 4 |
| linuxserver/nginx | 1 | 0 | 0 | 4 |

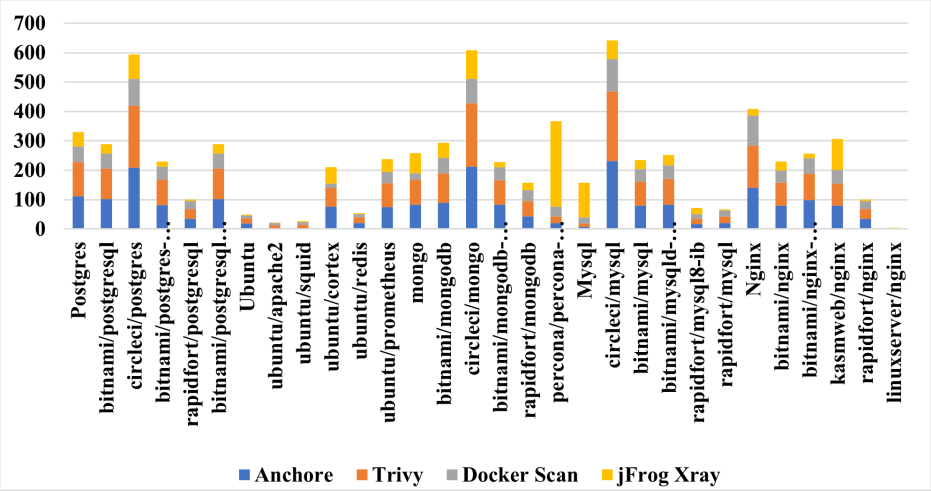



Figure 6.2: Total number of vulnerabilities reported by each tool

All the tools categorize the security vulnerabilities into four types based on severity levels: Critical, High, Medium, and Low. Table 6.8 and fig 6.3 shows the severity wise vulnerabilities in all the selected docker images. According to table 6.8, some of the images have zero critical vulnerabilities using all the selected tools and most of the images have higher low severity vulnerabilities (RQ2). From fig 6.3, we can say that linuxserver/nginx is more secure as compared to all the selected images as it has total 5 vulnerabilities using all the selected tools. We also analyze that official images are more secure as compared to their corresponding verified images.

Table 6.8: Severity wise vulnerabilities reported in each image

| **Image** | **Critical** | **High** | **Medium** | **Low** |
|---|---|---|---|---|
| Postgres | 4 | 60 | 27 | 239 |
| bitnami/postgresql | 2 | 35 | 20 | 232 |
| circleci/postgres | 68 | 162 | 118 | 245 |
| bitnami/postgres-exporter | 2 | 30 | 20 | 178 |
| rapidfort/postgresql | 0 | 5 | 6 | 89 |
| bitnami/postgresql-repmgr | 2 | 35 | 20 | 232 |
| Ubuntu | 0 | 0 | 16 | 42 |
| ubuntu/apache2 | 0 | 0 | 4 | 9 |
| ubuntu/squid | 0 | 0 | 7 | 20 |
| ubuntu/cortex | 3 | 101 | 76 | 31 |
| ubuntu/redis | 0 | 0 | 14 | 40 |

| | | | | |
|---|---|---|---|---|
| ubuntu/prometheus | 0 | 41 | 112 | 84 |
| mongo | 2 | 46 | 139 | 72 |
| bitnami/mongodb | 5 | 65 | 24 | 199 |
| circleci/mongo | 4 | 71 | 338 | 195 |
| bitnami/mongodb-exporter | 2 | 28 | 20 | 178 |
| rapidfort/mongodb | 3 | 40 | 58 | 106 |
| percona/percona-server-mongodb | 6 | 87 | 175 | 99 |
| Mysql | 2 | 44 | 74 | 37 |
| circleci/mysql | 61 | 155 | 94 | 332 |
| bitnami/mysql | 2 | 35 | 19 | 179 |
| bitnami/mysqld-exporter | 2 | 45 | 26 | 179 |
| rapidfort/mysql8-ib | 0 | 0 | 23 | 48 |
| rapidfort/mysql | 0 | 5 | 2 | 60 |
| Nginx | 8 | 40 | 89 | 271 |
| bitnami/nginx | 2 | 35 | 19 | 174 |
| bitnami/nginx-ingress-controller | 4 | 32 | 20 | 200 |
| kasmweb/nginx | 31 | 176 | 90 | 9 |
| rapidfort/nginx | 0 | 5 | 2 | 93 |
| linuxserver/nginx | 0 | 3 | 1 | 1 |

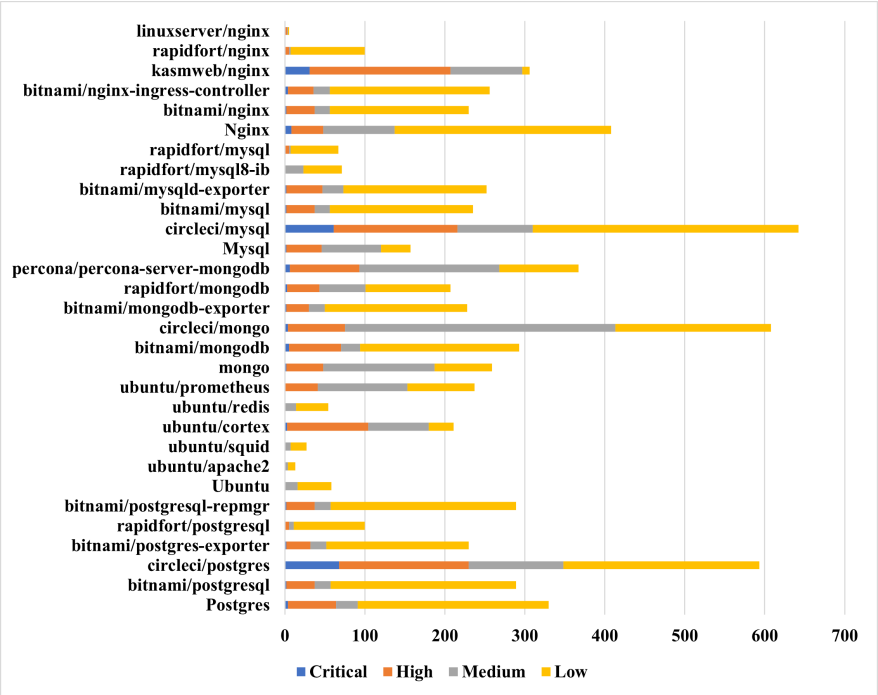



Figure 6.3: Severity wise vulnerabilities reported in each image

During scanning of the docker images, we identify that there are some vulnerabilities which occur most of the times in all the images. CVE-2022-1271, CVE-2005-2541, CVE-2022-0563, CVE-2019-9192, CVE-2017-11164, CVE-2022-29458, and CVE-2019-8457 are some of the most identified vulnerabilities in all the images using all the selected tools. We identify total 342 unique vulnerabilities in all the images. Fig 6.4 depicts the number of CVE ID published per year. Although most of the vulnerabilities identified are from 2022(155 total unique vulnerability), the oldest vulnerability identified is from 2005. Also, 14 vulnerabilities are published in 2023 till now (RQ3).

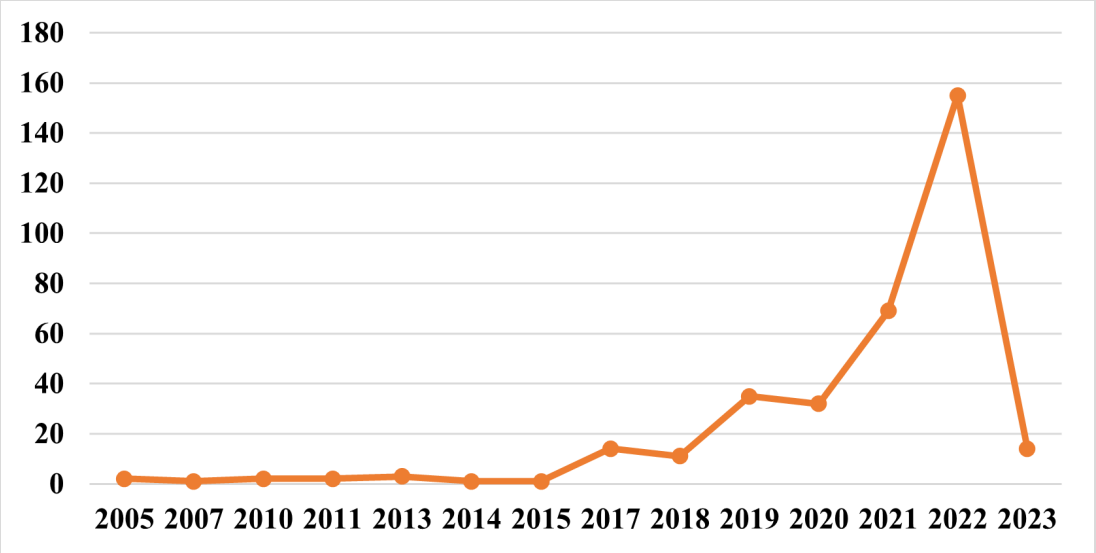



Figure 6.4: CVEs Published per Year

## 6.4 SDP with Optimization of DL Techniques using Hybrid Metaheuristic Approach

In this study, a hybrid metaheuristic-based hyperparameter tuned CNN model is proposed for SDP using six open-source Java-based datasets that include both process and product metrics. The model employs a hybrid optimization of the GWO and WOA to fine-tune critical hyperparameters such as epochs, learning rate, dropout rate, and batch size. The predictive performance of the proposed model is evaluated using standard performance metrics, including accuracy, AUC, F1-score, precision, recall, and MCC.

The results are compared against three baselines, CNN without HPT, CNN tuned with GWO, and CNN tuned with WOA to demonstrate the effectiveness of the hybrid optimization approach in enhancing model performance and generalization.

The following RQs are addressed in this study:

RQ1: Does adding HPT techniques improves the performance of CNN model in SDP problems?

RQ2: Do proposed hybrid model of GWO and WOA gives better performance as compared to CNN without HPT and individual GWO and WOA model in SDP problems?

RQ1 explores whether the inclusion of HPT can enhance the CNN model's predictive capability by fine-tuning critical parameters such as learning rate, epochs, batch size, and dropout rate, leading to better generalization and convergence. RQ2 focuses on assessing whether the hybrid optimization approach can leverage the exploration strength of GWO and the exploitation capability of WOA to achieve a more balanced search of the hyperparameter space. Together, these research questions aim to determine if the integration of metaheuristic optimization techniques can significantly improve the robustness, accuracy, and efficiency of CNN-based SDP models when compared to baseline and individually tuned counterparts.

### 6.4.1 Experimental Framework

Figure 6.5 shows the various steps followed to perform the experiment. In proposed model, a hybrid meta-heuristic optimization technique, which is a combination of GWO and WOA, is applied to optimize the hyperparameters of CNN for SDP. It comprises the best characteristic of both GWO and WOA. WOA struggles with performance during the exploitation phase and often becomes trapped in local optima. In contrast, GWO has shown strong performance in the exploitation phase, particularly when tested on unimodal benchmark functions, making it highly competitive among common metaheuristic algorithms. Thus, this paper aims to hybridize GWO with WOA to address these limitations, leveraging GWO's strengths in effectively exploiting optimal solutions. The experiment is conducted using 6 java-based open-source

datasets. The hyperparameters used in this study are epochs, dropout rate, batch size, and learning rate. The proposed model's prediction performance is examined in respect of accuracy, precision, AUC, f1 score, MCC, and recall.

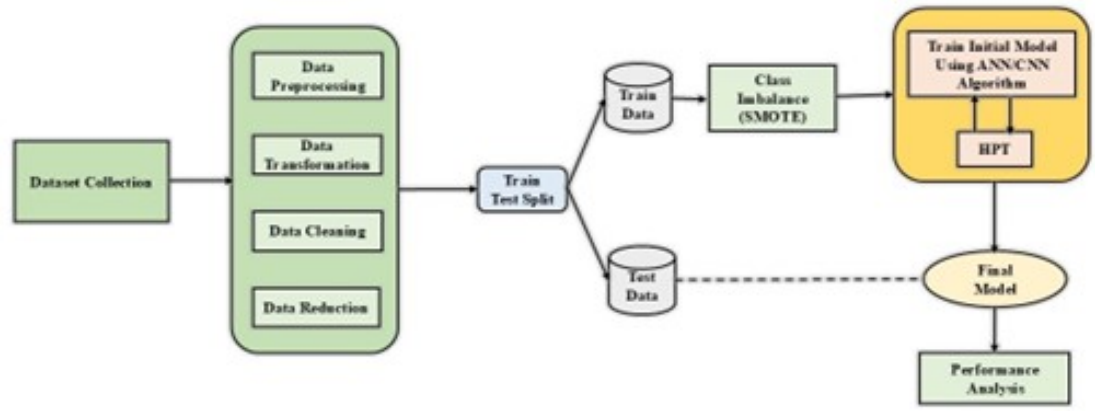



Figure 6.5: Proposed Architecture

### 6.4.2 Result Analysis

In this section, experimental results of this study are discussed. The performance of the proposed model is examined on 6 datasets from open-source repository. Note that, we have used accuracy, AUC, f1 score, MCC, precision, and recall measures to examine the prediction performance of proposed model. The preprocessed dataset is splitted into training and testing dataset for the purpose of training the hyperparameter tuned CNN model and testing the prediction effectiveness of the model. In this study, hybrid model of WOA and GWO is used to improve the hyperparameters of CNN algorithm.

In this study, Google Colab is utilized to train and assess the performance of DL models. Tensorflow and Keras frameworks are used to generate the models. The server was running on a machine having x64 based processor and 192 GB RAM. The language used to write the code is python 3.8. Python, keras, tensorflow, and scikit-learn libraries are used for developing the models.

Table 6.9 displays the experimental outcomes of the proposed model in respect of accuracy, AUC, f1 score, MCC, precision, and recall. With respect to table 6.9, the

Table 6.9: Table

| | **Accuracy (%)** | **Precision** | **Recall** | **F1 score** | **AUC** | **MCC** |
|---|---|---|---|---|---|---|
| **Ant 1.6** | 90.6 | 0.681 | 0.914 | 0.781 | 0.896 | 0.719 |
| **Ant 1.7** | 87.2 | 0.569 | 0.804 | 0.667 | 0.845 | 0.604 |
| **Xalan 2.5.0** | 71.9 | 0.567 | 0.932 | 0.705 | 0.745 | 0.489 |
| **Xalan 2.6.0** | 84.2 | 0.753 | 0.887 | 0.776 | 0.835 | 0.655 |
| **Xerces 1.3.0** | 84.4 | 0.742 | 0.824 | 0.623 | 0.836 | 0.658 |
| **Xerces 1.4.4** | 96.3 | 0.957 | 0.989 | 0.973 | 0.948 | 0.914 |

answer to RQ1 is discussed.

- · *RQ1: Does adding HPT techniques improves the performance of CNN model in SDP problems?*

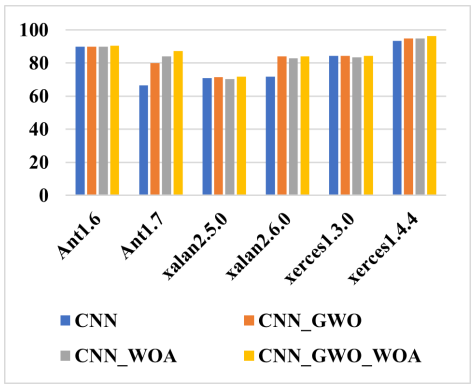
According to table 6.9, we analyze that the proposed model achieves maximum accuracy of 96.3% for xerces 1.4.4 dataset and accuracy lie between 84%-97% for all the datasets except xalan 2.5.0 datasets. AUC and MCC scores for proposed model lies between 0.830-0.950 and 0.6-0.92 for all the datasets except xalan 2.5.0, which indicates that it can successfully predict the defects in the software module.

Figure 6.6a, 6.6b, 6.6c, 6.6d, 6.6e, and 6.6f shows the comparative analysis of results of the proposed model with CNN without HPT, CNN_GWO, and CNN_WOA in respect of accuracy, AUC, MCC, precision, f1 score, and recall Based on figure 6.6, the answer to RQ2 is discussed.

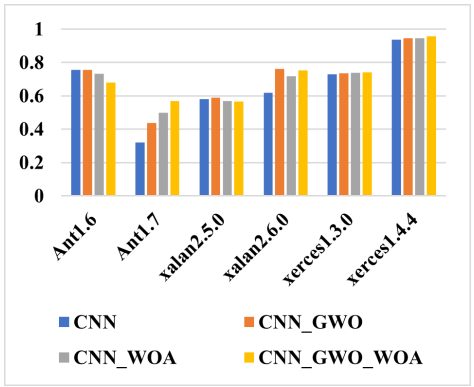
- *RQ2: Do proposed hybrid model of GWO and WOA gives better performance as compared to CNN model without HPT, and individual GWO and WOA model in SDP problems?*

Figure 6.6a shows the comparative analysis of the proposed model's prediction performance with CNN without HPT, CNN_GWO, and CNN_WOA, evaluated in terms of accuracy. In case of CNN without HPT, accuracy lies in the range of 60% - 93.3%, in case of CNN_GWO, accuracy lies in the range of 71.4% - 94.8%, in case of CNN_WOA, accuracy lies in the range of 70.4% - 94.8%, and in case of CNN_GWO_WOA, accuracy lies in the range of 71.9% - 96.3%. Figure 6.6b shows the comparative analysis

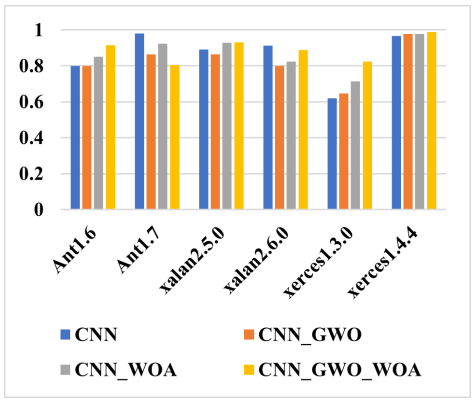
of the proposed model's prediction performance with CNN without HPT, CNN_GWO, and CNN_WOA, evaluated in terms of precision. In case of CNN without HPT, precision lies in the range of 0.321 – 0.937, in case of CNN_GWO, precision lies in the range of 0.436 – 0.947, in case of CNN_WOA, precision lies in the range of 0.5 – 0.947, and in case of CNN_GWO_WOA, precision score falls between 0.567 – 0.957. Figure 6.6c shows the comparative analysis of the proposed model's prediction performance with CNN without HPT, CNN_GWO, and CNN_WOA, evaluated in terms of recall. In case of CNN without HPT, recall lies in the range of 0.619 – 0.967, in case of CNN_GWO, recall lies in the range of 0.647 – 0.978, in case of CNN_WOA, recall lies in the range of 0.714 – 0.978, and in case of CNN_GWO_WOA, recall score falls between 0.804 – 0.989. Figure 6.6d shows the comparative analysis of the proposed model's prediction performance with CNN without HPT, CNN_GWO, and CNN_WOA, evaluated in terms of f1 score. In case of CNN without HPT, f1 score falls between 0.484 – 0.952, in case of CNN_GWO, f1 score falls between 0.565 – 0.963, in case of CNN_WOA, f1 score falls between 0.615 – 0.962, and in case of CNN_GWO_WOA, f1 score falls between 0.623 – 0.973. Figure 6.6e shows the comparative analysis of the proposed model's prediction performance with CNN without HPT, CNN_GWO, and CNN_WOA, evaluated in terms of AUC. In case of CNN without HPT, AUC score falls between 0.739 – 0.914, in case of CNN_GWO, AUC score falls between 0.742 – 0.931, in case of CNN_WOA, AUC score falls between 0.741 – 0.930, and in case of CNN_GWO_WOA, AUC score falls between 0.745 – 0.948. Figure 6.6f shows the comparative analysis of the proposed model's prediction performance with CNN without HPT, CNN_GWO, and CNN_WOA, evaluated in terms of MCC. In case of CNN without HPT, f1 MCC lies in the range of 0.422 – 0.845, in case of CNN_GWO, MCC lies in the range of 0.475 – 0.879, in case of CNN_WOA, MCC lies in the range of 0.452 – 0.879, and in case of CNN_GWO_WOA, MCC lies in the range of 0.489 – 0.914. So, from the figure 6.6, we can conclude that there is a noticeable improvement in the performance of proposed model's predictive accuracy as compared to CNN without HPT, CNN_GWO, and CNN_WOA models, highlighting the beneficial effects of HPT and hybrid metaheuristic algorithm.



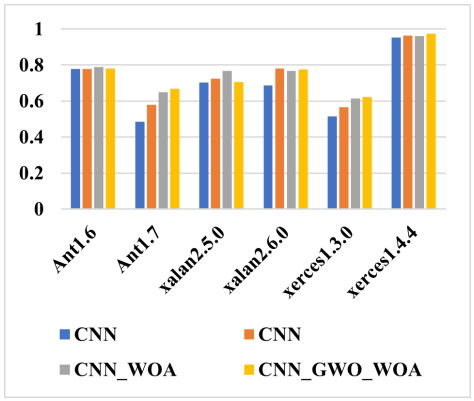

(a) Accuracy values



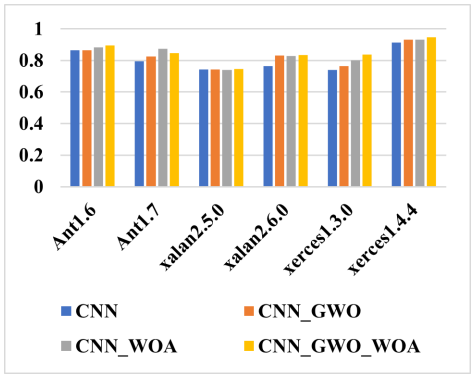

(b) Precision values



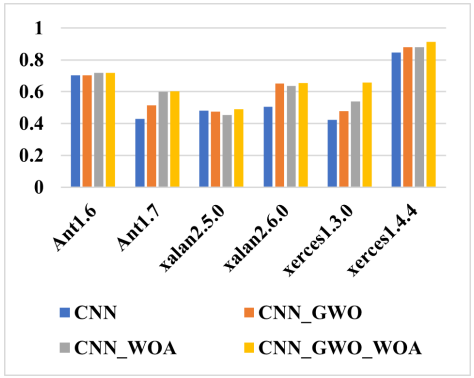

(c) Recall Values




(d) F1 score Values




(e) AUC values




(f) MCC values

Figure 6.6: Comparison of proposed model with CNN, CNN_GWO, and CNN_WOA models

- Statistical Analysis

In the previous sections, we have discussed the experimental results of the proposed model on 6 java-based datasets and we found that the proposed model outperforms in-

dividual GWO, WOA, and CNN without HPT. Furthermore, friedman test is also used to authenticate the prediction efficacy of the proposed model over individual GWO, WOA, and CNN without HPT. Table 6.10 shows the mean ranks based on accuracy and AUC measure. According to table 6.10, the proposed model is assigned with highest mean rank.

Table 6.10: Mean ranks based on accuracy and AUC values

| **Technique Name** | **Mean rank based on accuracy** | **Mean rank based on AUC** |
|---|---|---|
| CNN | 1.5 | 1.33 |
| CNN_GWO | 2 | 2.5 |
| CNN_WOA | 2.67 | 2.33 |
| CNN_GWO_WOA | 3.83 | 3.83 |

To justify the null hypothesis, we compare the calculated value of friedman statistics with the chi square tabulated value at degree of freedom (df) k-1, where k represents number of techniques. The calculated value of the friedman test based on accuracy is 12.22 and based on AUC is 11.4. It can be observed from chi square table, that at k-1 df, the friedman statistic value at 0.05 significant is 7.815 which is less than calculated values. Therefore, we reject the null hypothesis. Hence, we may deduce that there exists statistically notable difference between the performance of the proposed model and other models. So, nemenyi post hoc test is applied to assess the effectiveness of the proposed model. It returns pairwise p-value between two models. The result illustrates that there exists a notable difference between the performance of CNN and CNN_GWO_WOA. The performance of all other models does not differ significantly.

## 6.5 Security and Performance Analysis of Dockerized FL Model for SDP

With the growing adoption of containerization in modern software engineering, deploying ML and DL models through Docker has become increasingly common due to its scalability, portability, and efficiency. This study develops and deploys a FL based SDP model within a Dockerized environment to ensure secure, resource-efficient, and

globally accessible model execution. Although DL frameworks offer powerful predictive capabilities, they may also introduce security vulnerabilities arising from outdated libraries or misconfigurations. Hence, this work not only evaluates the predictive performance of FL-based models but also conducts a comprehensive security and resource utilization analysis using various open-source tools.

The following RQs are addressed in this study:

RQ1: What is the difference between the performances of the FL model having CNN as the base model and the FL model having ANN as the base model?

RQ2: How vulnerable is the FL model?

RQ3: How does the FL model perform in terms of CPU utilization, memory utilization, network traffic usage, and disk usage?

RQ1 examines the prediction performance of FL with CNN as base model and FL with ANN as base model. The goal is to analyze which architecture offers better accuracy, generalization, and robustness within the FL framework. RQ2 is related to analyzing the security vulnerabilities in the generated FL model. With the help of various open source tools such as anchore, aqua trivy, jFrog xray, docker scout, and docker scan tools, the study evaluates the severity and types of vulnerabilities that may compromise the model's integrity and deployment security. RQ3 is related to analyzing the resource usage by generated FL model. By employing monitoring tools such as Prometheus, Grafana, and cAdvisor, the study measures and analyzes system-level metrics to assess the computational cost and scalability of the deployed models. Together, these RQs provide a comprehensive understanding of the predictive accuracy, security resilience, and resource utilization of federated learning models in containerized environments.

### 6.5.1 Experimental Framework

Figure 6.7 describes various steps followed to perform the experiment. This experimental setup presents a streamlined approach to FL implementation, starting with data

collection and preprocessing steps including train-test split and data sampling. The workflow demonstrates client-server data distribution where client data is distributed across multiple nodes (C1, C2,...,C10) while server data is used to initialize CNN/ANN algorithms. The FL process involves local model training at each client, followed by model aggregation at the central server to produce a unified model. The aggregated model is converted to HDF5 format and deployed through Flask API integration on Docker containers. The deployment is followed by parallel security analysis and performance evaluation processes to assess both the vulnerability landscape and resource utilization of the containerized FL system.

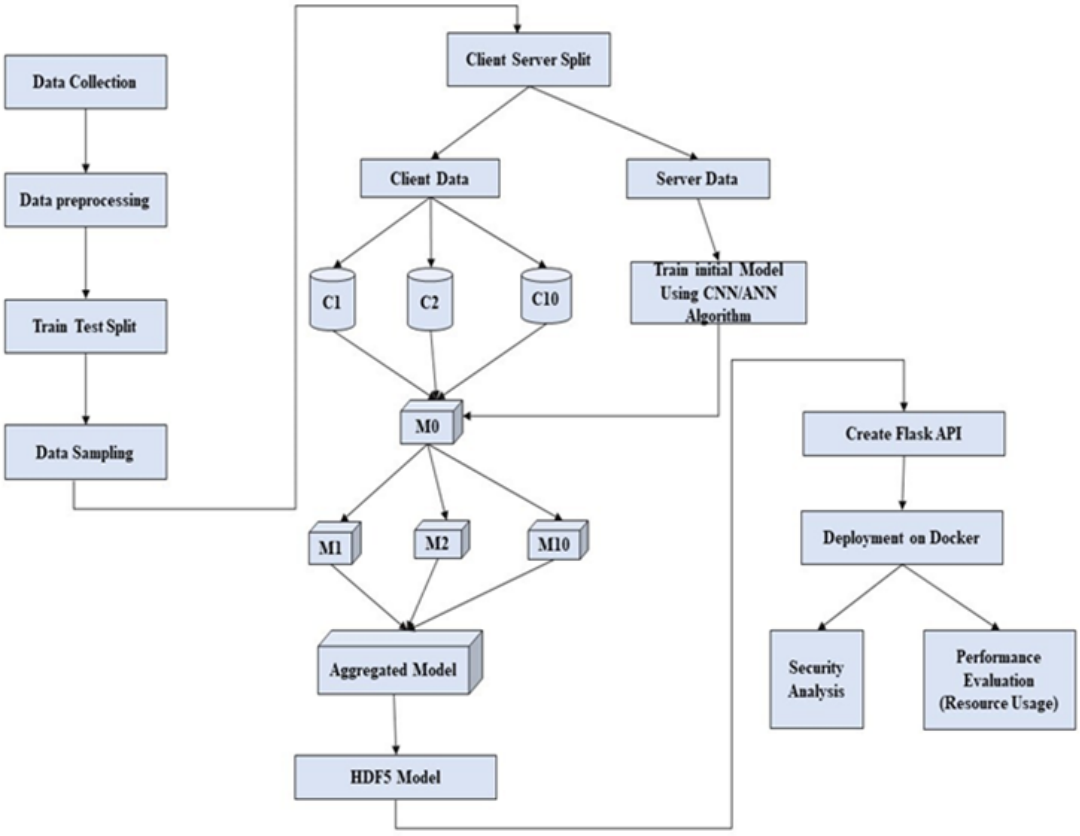



Figure 6.7: Proposed Architecture

### 6.5.2 Result Analysis

In this section, the experimental results are described. Firstly, the FL model is evaluated on 8 datasets. Then, we analyze the results based on performance metrics. Note that AUC and MCC measures are used to evaluate the prediction performance of the model. After this, we generate the flask api for the model and deploy it using docker. Finally, we analyze the security vulnerabilities and performance of the model with the help of open-source tools.

Table 6.11 represent the experimental results of the FL model in terms of AUC and MCC. According to table 6.11, the datasets ant 1.6, xalan 2.5.0, xalan 2.6.0, xerces 1.3.0, and xerces 1.4.4 have the best AUC and MCC values when using the FL model with CNN as the base model. Therefore, out of the 8 selected datasets, 5 of them show the best result when using CNN. Additionally, among all the selected datasets, xerces 1.4.4 shows the best AUC and MCC values for both FL models, whether it is using CNN as the base model or ANN as the base model. Based on table 6.11, the answer to RQ1 is presented.

- *RQ1: What is the difference between the performances of the FL model having CNN as the base model and the FL model having ANN as the base model?*

According to table 6.11, we observe that the FL model having CNN as the base model outperforms the FL model having ANN as the base model as in most of the cases CNN has high AUC and MCC values ANN. From table 6.11, we can also analyze that the FL model having CNN as the base model has high AUC and MCC values for the Xerces1.4.4 dataset. So, the model having CNN as the base model is deployed on docker to analyze the security vulnerabilities and to evaluate the performance in terms of resource usage. Note that we have used Anchore, jFrog Xray, Aqua Trivy, Docker Scan, and Docker Scout to analyze the vulnerabilities. All the tools categorize the security vulnerabilities into four types based on severity levels: Critical, High, Medium, and Low. Table 6.12 show the vulnerability results of all the chosen tools.

Table 6.11: AUC and MCC values for CNN and ANN

| **Project Name** | **CNN** | | **ANN** | |
|---|---|---|---|---|
| | **AUC** | **MCC** | **AUC** | **MCC** |
| Ant 1.6 | **0.836** | **0.608** | 0.696 | 0.386 |
| Ant 1.7 | 0.707 | 0.385 | **0.775** | **0.513** |
| jEdit 4.1 | 0.752 | 0.507 | **0.873** | **0.691** |
| Xalan 2.5.0 | **0.738** | **0.481** | 0.733 | 0.466 |
| Xalan 2.6.0 | **0.787** | **0.576** | 0.761 | 0.544 |
| Xerces 1.2.0 | 0.724 | 0.359 | **0.838** | **0.429** |
| Xerces 1.3.0 | **0.873** | **0.663** | 0.769 | 0.495 |
| Xerces 1.4.4 | **0.956** | **0.907** | 0.883 | 0.761 |

Anchore detects 313 vulnerabilities, Aqua Trivy detects 308 vulnerabilities, Docker Scout detects 180 vulnerabilities, Docker Scan detects 54 vulnerabilities, and jFrog Xray detects 33 vulnerabilities. Based on table 6.12, the answer to RQ2 is presented.

- *RQ2: How vulnerable is the FL model?*

According to table 6.12, docker image of the FL model having CNN as the base model is secure as there exists a smaller number of critical and high-severity vulnerabilities in the case of all the selected tools.

Table 6.12: Severity wise total number of vulnerabilities identified

| **Tools** | **Critical** | **High** | **Medium** | **Low** |
|---|---|---|---|---|
| Anchore | 0 | 7 | 194 | 112 |
| Jfrog Xray | 0 | 8 | 15 | 10 |
| Aqua Trivy | 0 | 5 | 189 | 114 |
| Docker Scan | 0 | 1 | 31 | 22 |
| Docker Scout | 1 | 7 | 113 | 59 |

During the scanning of docker images, we identified that there are some vulnerabilities that occur frequently in all the tools. Table 6.13 represents information related to some of the frequently occurring vulnerabilities in all the tools. These vulnerabilities can lead to a chance of denial-of-service attack, high resource consumption, loss of confidentiality, leakage of sensitive information, etc.

Table 6.13: Identified security vulnerabilities

| S.No | CVE-ID | Severity | Description |
|---|---|---|---|
| 1 | CVE-2023-31484 | Medium | Improper certificate validation |
| 2 | CVE-2023-29491 | Medium | Out of bounds write |
| 3 | CVE-2023-46136 | Medium | Uncontrolled resource consumption leading to denial of service |
| 4 | CVE-2022-47696 | Medium | Segmentation fault leading to denial of service or other unspecified impacts |
| 5 | CVE-2023-0459 | Medium | Allows to leak kernel information |
| 6 | CVE-2023-32731 | High | Sensitive information disclosure |
| 7 | CVE-2023-20588 | High | Loss of confidentiality due to division by zero |
| 8 | CVE-2023-4206 | High | Use after free |
| 9 | CVE-2023-42753 | High | Potential slab out of bound access due to integer underflow |
| 10 | CVE-2023-42756 | High | Race condition |
| 11 | CVE-2023-28322 | Low | Information disclosure |
| 12 | CVE-2023-38546 | Low | Cookie injection with none file |
| 13 | CVE-2023-2602 | Low | Memory leak |
| 14 | CVE-2021-33294 | Low | An infinite loop leading to denial of service |
| 15 | CVE-2021-39537 | Low | Heap based buffer overflow |

Table 6.14: Number of users and spawn rate

| Number of Runs | Number of Users | Spawn Rate |
|---|---|---|
| Run 1 | 200 | 50 |
| Run 2 | 500 | 50 |
| Run 3 | 1000 | 50 |
| Run 4 | 2000 | 50 |
| Run 5 | 2000 | 100 |
| Run 6 | 1000 | 100 |

A load test is conducted on the flask API of the ML model using the locust tool. 6

runs on the locust tool is performed to check the performance. Table 6.14 shows the details about number of users and spawn rate taken during various runs. The server is running on a machine with 192 GB RAM and x64-based processor. Figure 6.8 shows the results obtained after performing the load test. During the load test initially, we input the number of users and spawn rate. Spawn rate means the speed at which users are created until the given number of users are created.

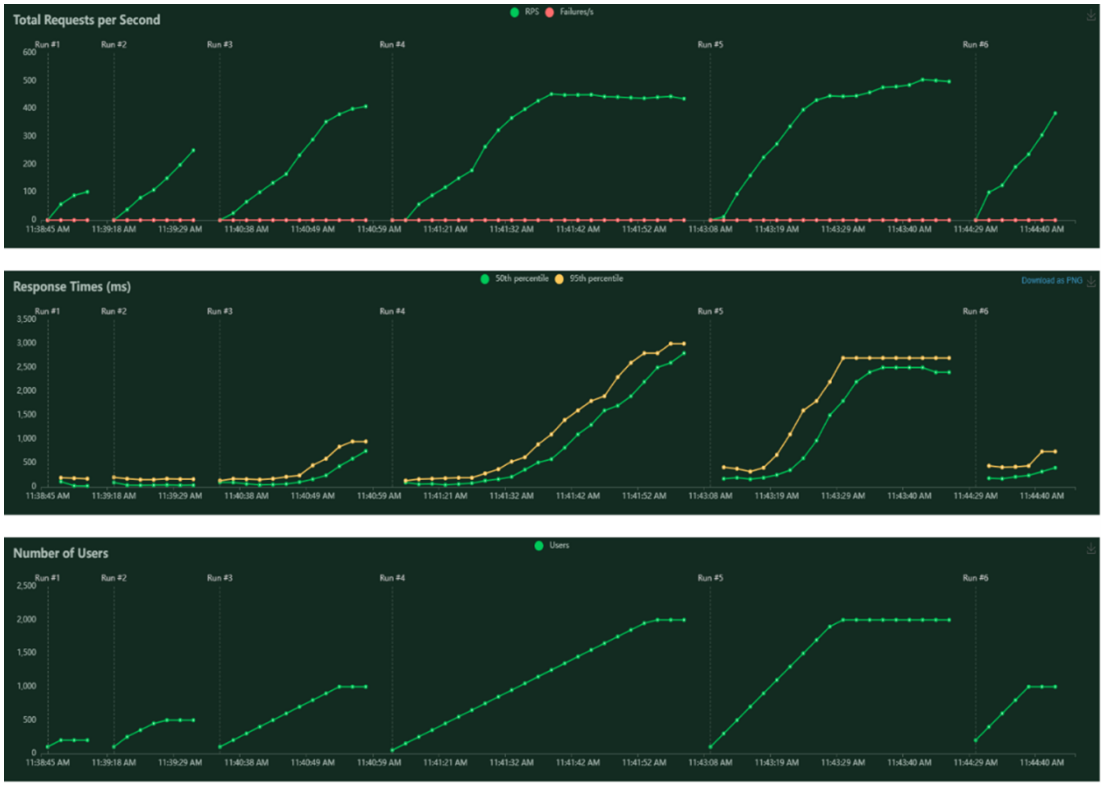



Figure 6.8: Load testing using locust

The metrics which are measured during the load test are response time, number of users, and total requests per second. From figure 6.8, it can be analyzed that as the number of users increases with a constant spawn rate of 50 users the response rate increases but when the spawn rate changes from 50 to 100 users per second then the response rate decreases. In figure 6.8, after run 4 response rate decreases and becomes constant after a certain amount of time. From figure 6.8, we can also analyze that in the case of run 4 and run 5, and run 3 and run 6 number of users are same but there is a change in spawn rate so when we increase the spawn rate, the response rate decreases

during run 5 and run 6.

Note that in this study, open-source tools such as cAdvisor, Prometheus, Grafana, sematext, and docker stats have been used to monitor the performance of docker container. A comprehensive monitoring and visualization infrastructure is established by integrating cAdvisor, Prometheus, and Grafana. Container resource monitoring is conducted using cAdvisor, an open-source tool designed for collecting and analyzing resource utilization metrics of containerized applications. cAdvisor is integrated into docker desktop to monitor CPU utilization, memory utilization, network traffic, and disk usage of the deployed containers. We use Prometheus to collect and store time-series data from our experimental environment. Prometheus continuously receives data from cAdvisor and stores it. To visualize the data which is stored by Prometheus, we employed Grafana, an open-source data visualization tool.

Over a monitoring period of 6 hours, various metrics including CPU utilization, memory consumption, network traffic, and data usage are collected. Data was collected every 10s. Table 6.15 shows the resource usage (CPU utilization, memory utilization, network traffic usage, and disk usage) of FL using five monitoring tools. Among all the selected tools, Grafana shows high resource usage with CPU usage at 85%, memory usage at 453 TB, and network traffic at 1.50 MB/s over a monitoring period of 6 hours. cAdvisor, Grafana, Prometheus, and docker stats are specifically used for container monitoring. Sematext provides a wide range of observability tools that covers logging, monitoring, and tracing for both applications and infrastructure. However, docker stats does not continuously monitor the container resource usage; it only provides an immediate snapshot of resource usage by the container.

Table 6.15: Resource usage using monitoring tools

| **Tools** | **CPU usage** | **Memory usage** | **Network traffic** | **Disk usage** |
|---|---|---|---|---|
| cAdvisor | 80 | 928.72 MB | 317 KB/2389.4 KB | 539.81 MB |
| Docker Stats | 146.68 | 685.4 MB | 12.7 KB/53.7 KB | 0B/0B |
| Sematext | 137.4 | 618.6 MB | 104.4 KB/2 MB | 0B/0B |
| Prometheus | 80 | 453.20 MB | 1.50 KB/s | 108 KB |
| Grafana | 85 | 453 TB | 1.50 MB/s | - |

According to table 6.15, cAdvisor, Prometheus, and Grafana shows high resource usage as compared to docker stats and sematext.

Figure 6.9 illustrates the CPU utilization trend and figure 6.10 illustrates the memory usage trend using Prometheus. Figure 6.11 illustrates the network traffic trend and figure 6.12 showcases the total bytes used. In figure 6.9, y-axis represents CPU usage percentage, in figure 6.10, y-axis represents memory usage in megabytes, in figure 6.11, y-axis represents bandwidth usage in kilobytes per second, and in figure 6.12, y-axis represents bytes used in kilo bytes. In figure 6.9, 6.10, 6.11 and 6.12, X-axis represents time frame. The results show that after 6 hours of run, CPU usage is 80%, memory usage is 453.20 MB, network traffic is 1.50 KB/s, and total bytes used are approximately 108KB.

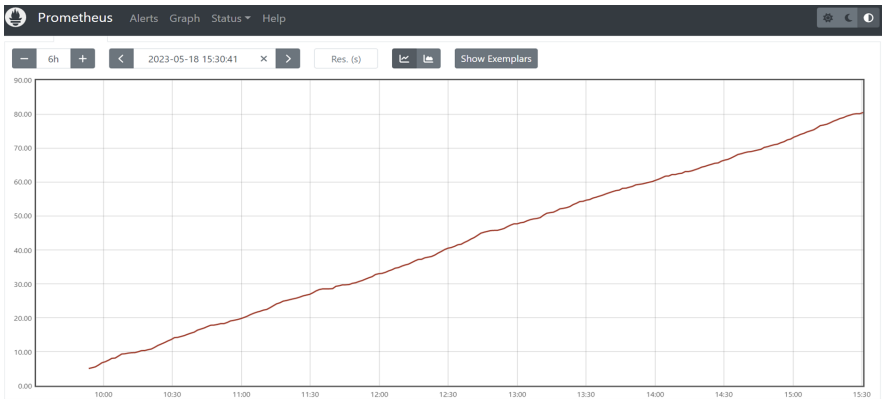



Figure 6.9: Prometheus CPU usage




Figure 6.10: Prometheus memory usage

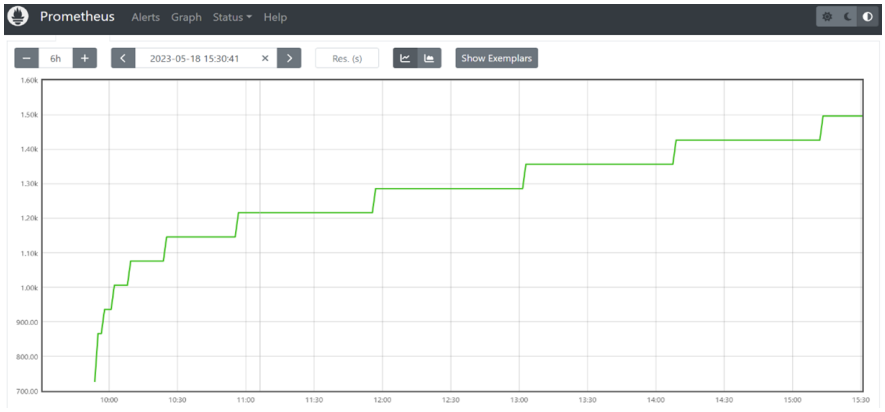



Figure 6.11: Prometheus network traffic

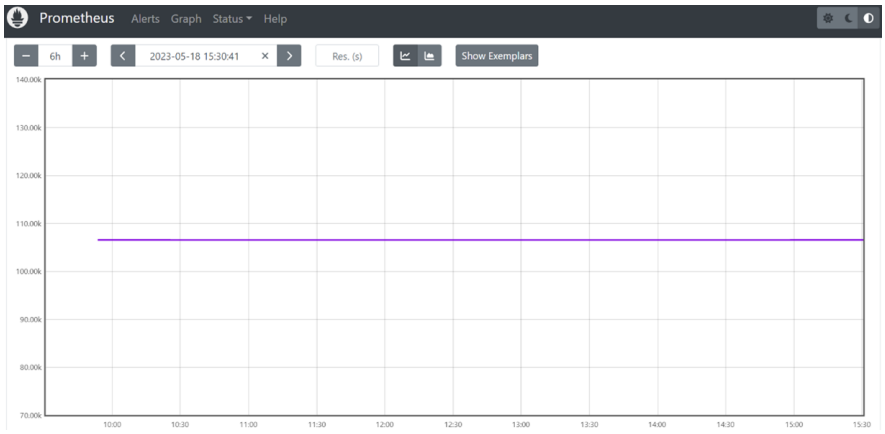



Figure 6.12: Prometheus total bytes used

Figure 6.13 illustrates the CPU utilization trend, figure 6.14 illustrates the memory utilization trend, and figure 6.15 showcases the network traffic trend using Grafana dashboards. In figure 6.13, y-axis represents CPU usage percentage, in figure 6.14, y-axis represents memory usage in tera bytes, and in figure 6.15, y-axis represents bandwidth usage in megabytes per second. X- axis represents time frame in figure 6.13, 6.14, and 6.15. The result shows that after 6 hours of run, CPU usage is approximately 85%, memory usage is 453 TB, and network traffic is 1.50 MB/s.

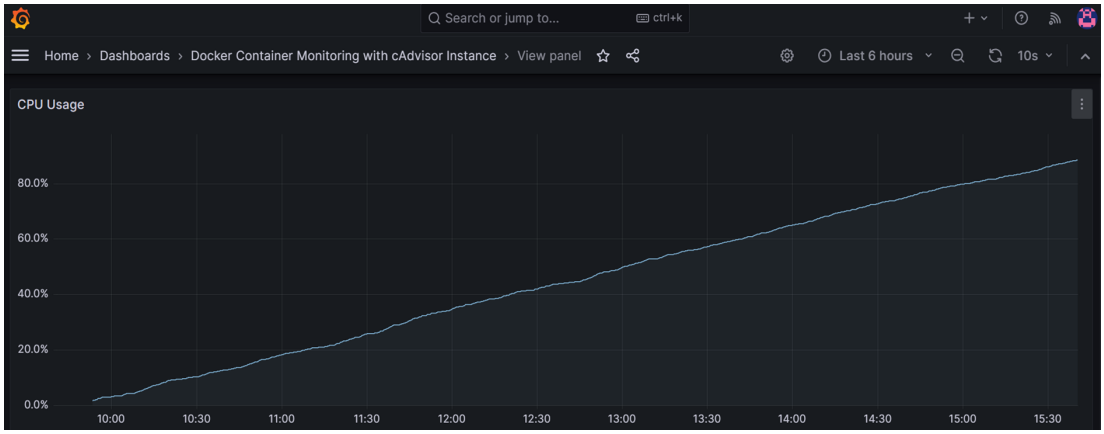



Figure 6.13: CPU usage using Grafana dashboard




Figure 6.14: Memory usage using Grafana dashboard

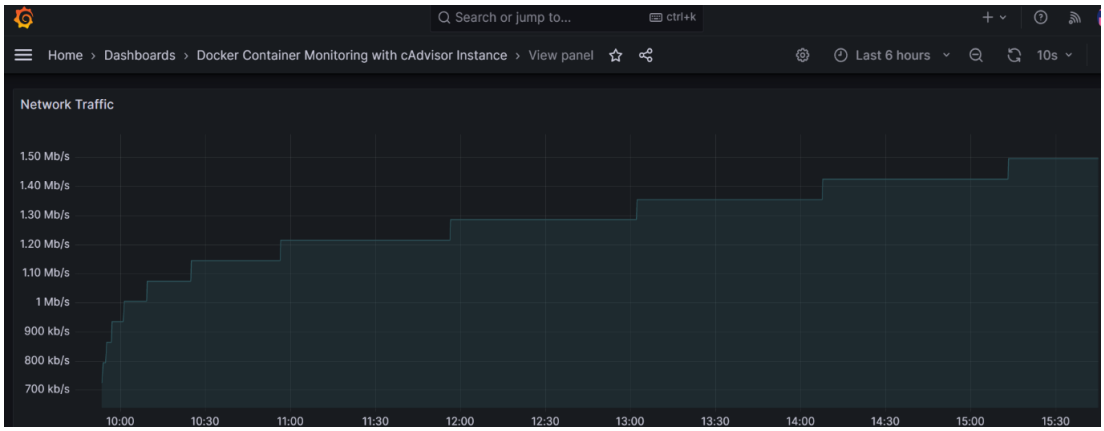



Figure 6.15: Network traffic using Grafana dashboard

- · *RQ3: How does the FL model perform in terms of CPU utilization, memory utilization, network traffic usage, and disk usage?*

According to table 6.14 and figure 6.8, it can be analyzed that the response rate increases when the spawn rate is 50 users but when the spawn rate increases from 50 users to 100 users then the response rate decreases and it can handle multiple users request simultaneously. According to table 6.15, figure 6.9, figure 6.10, figure 6.11, figure 6.12, figure 6.13, figure 6.14, and figure 6.15, we can analyze that it performs efficiently as it consumes fewer resources.

The results of this study have implications for practitioners and researchers. The summary of the main findings of this study is as follows: RQ1 examines the prediction performance of FL with CNN as base model and FL with ANN as base model. Based on table 6.11, it is observed that FL with CNN as base model achieves higher prediction performance compared to FL with ANN as base model. This is evident from its high AUC and MCC values for most of the selected datasets. RQ2 investigates the security vulnerabilities in the generated model using open-source tools. Based on table 6.12, it is observed that FL model with CNN as base model is deemed secure. This is evident from its lower count of critical and high severity vulnerabilities across most of the selected tools. RQ3 examines the performance and resource usage of generated model in terms of CPU utilization, memory utilization, network traffic usage, and disk usage using open-source tools. Based on table 6.14 and figure 6.8, it is observed that as the spawn rate increases from 50 to 100 with a constant number of users, the response rate decreases but the model can handle multiple requests simultaneously. Additionally, based on table 6.15 and figures 6.9, 6.10, 6.11, 6.12, 6.13, 6.14, 6.15, it is observed that after 6 hours of running, the model consumes fewer resources.

Even though the FL model seems promising, further research is advised to confirm its superiority and investigate possible refinements. This study provides a significant contribution to the field of SDP and establishes a framework for further research into the efficacy of the FL model and other techniques.

## 6.6 Security Analysis of Optimized FL Model for SDP

With the growing adoption of FL in software analytics, ensuring the security and reliability of deployed models has become increasingly critical. While FL enhances data privacy by enabling decentralized model training, its integration with DL architectures and deployment through containerized environments introduces new security vulnerabilities that can compromise model integrity and system safety. These vulnerabilities may arise from outdated libraries, misconfigurations, insecure dependencies within container images, or supply chain attacks targeting ML frameworks.

In this study, we conduct a comprehensive security analysis of optimized FL models for SDP to assess their resilience against potential threats in a containerized deployment environment. The optimized FL models, FLCNN and FLANN, both enhanced with HPT using the GWO are deployed as Docker microservices to ensure modularity, scalability, and ease of integration into production workflows. To evaluate the security posture of these deployments, we employ multiple open-source container vulnerability scanning tools, including Anchore, Aqua Trivy, Clair, JFrog Xray, and Snyk, to detect and categorize vulnerabilities across severity levels (critical, high, medium, and low). By integrating model optimization with deployment-level security evaluation, this study contributes toward developing robust, privacy-preserving, and secure predictive frameworks suitable for modern software engineering environments

The following RQs are addressed in this study:

RQ1: What are the performance differences between FL models using CNN and ANN base architectures when enhanced with HPT?

RQ2: How does HPT impact the prediction accuracy of FL models?

RQ3: What security vulnerabilities are present in the Dockerized FL models, and how can they be identified?

Based on these RQs, the key contributions of this research are as follows:

- FL for SDP: Design and implementation of two privacy-preserving FL models,

FLCNN and FLANN, optimized with GWO based HPT for SDP.

- Comprehensive Evaluation: Empirical comparison of the proposed models across eight public Java projects using AUC and MCC, supported by statistical validation of observed differences.

- Secure Deployment Analysis: Containerization of the best FL model using Docker and systematic vulnerability assessment with multiple scanning tools (Anchore, Aqua Trivy, JFrog Xray, Snyk).

- Quantitative Mitigation Study: Categorization of vulnerabilities by severity, proposal of remediation steps, and evaluation of before/after improvements in container security.

- Integrated Perspective: Establishment of a unified framework linking model accuracy, federated privacy, and deployment security, providing actionable insights for secure and scalable adoption of FL based SBP systems in industrial settings.

### 6.6.1 Experimental Framework

Figure 6.16 shows the experimental setup to perform the experiment. The experimental setup follows a comprehensive FL pipeline beginning with dataset collection and preprocessing stages including data transformation, cleaning, and reduction to address class imbalance. The preprocessed data undergoes train-test split followed by client-server distribution, where data is partitioned among multiple clients (C1, C2, ..., C10) while maintaining a central server dataset. Each client trains local models that are aggregated into a global model (M0, M1, M2, ..., M10) through federated averaging, with HPT integrated using GWO to optimize model performance. The final aggregated model is saved in HDF5 format and deployed via Flask API on Docker containers. Post-deployment, security analysis using open-source scanning tools is conducted to evaluate the system's robustness and efficiency.

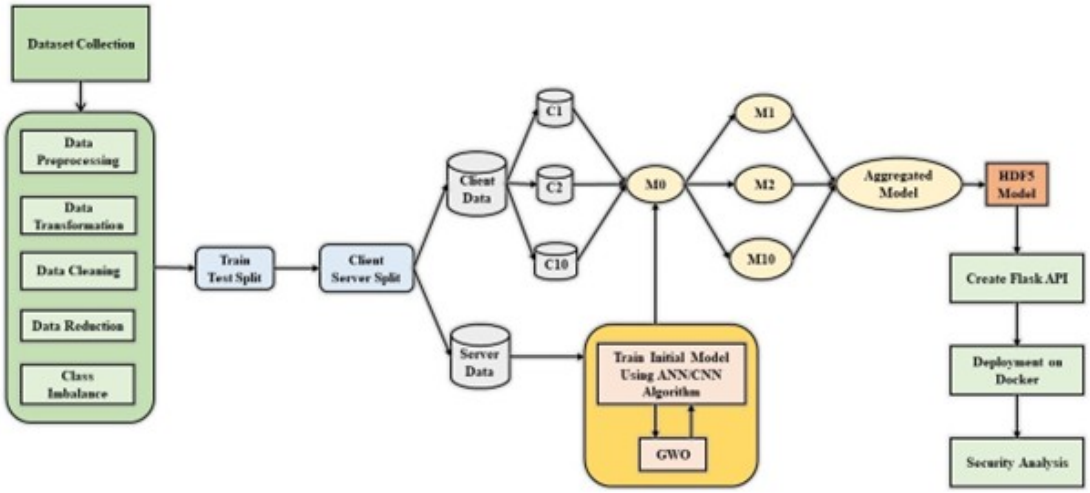



Figure 6.16: Proposed Architecture

**Experimental setup**

- The FL experiments were executed on the Google Colab platform, utilizing a server configured with an Intel Xeon x64-based processor, Tesla T4 GPU, and 192 GB RAM. The training framework implemented TensorFlow 2.12 and Keras 2.13 under Python 3.10. Although precise CPU and GPU specifications, including clock speeds and power consumption, were not further detailed, this setup reflects a commonly accessible environment for contemporary ML research.
- The FL setup consisted of 10 clients performing model training over 20 global communication rounds. Local training was performed across varying epoch counts (5, 10, 20, 50) to analyze the effect on model performance and computational overhead.
- The CNN-based model architecture (FLCNN) comprised multiple convolutional layers with varying filter sizes, interspersed with max pooling, dropout, and batch normalization, followed by several fully connected dense layers. The ANN-based model (FLANN) utilized a sequence of fully connected dense layers with ReLU activation and dropout regularization.
- Based on experimental logs, the CNN model training averaged approximately

137 seconds per session, while the ANN model completed training in around 88 seconds per session. Model file sizes were roughly estimated at 25 MB for the CNN and 18 MB for the ANN, contributing to total communication costs ranging from 7.2 GB to 10 GB for the entire FL process, accounting for client-server synchronization transmission in each round.

### 6.6.2 Result Analysis

This section presents and discusses the experimental results of the proposed study. The performance of the FL models is evaluated on eight datasets sourced from open-source repositories. Prediction effectiveness is measured using two standard metrics: AUC and MCC. To optimize the hyperparameters of both CNN and ANN models, the GWO algorithm is employed. The resulting models are deployed as Flask APIs within Docker containers to facilitate scalable, portable deployment. Furthermore, the security posture of these containerized models is rigorously assessed using multiple open-source container scanning tools, including Anchore, Aqua Trivy, Snyk, and JFrog Xray.

Table 6.16 describes the performance comparison of FLCNN with HPT and FLANN with HPT in terms of AUC and MCC. The answer to RQ1 is described based on Table 6.16.

- · *RQ1: What are the performance difference between FL models using CNN and ANN base architectures when enhanced with HPT?*

According to Table 6.16, all the datasets, except jedit4.1, have better AUC and MCC values for FLCNN with HPT compared to the FLANN with HPT model. Furthermore, the dataset xerces1.4.4 has better AUC and MCC values for both FLCNN with HPT (0.961 and 0.917) and FLANN with HPT (0.885 and 0.768) models compared to the other selected datasets. We can conclude that FLCNN with HPT model outperforms FLANN with HPT model in terms of both AUC and MCC values.

Table 6.16: Performance comparison of FLCNN with HPT and FLANN with HPT

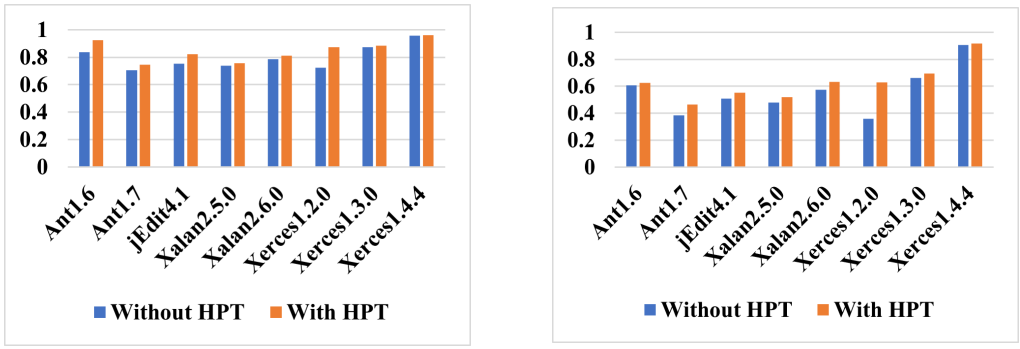
| **Dataset Name** | **FLCNN with HPT** | | **FLANN with HPT** | |
|---|---|---|---|---|
| | **AUC** | **MCC** | **AUC** | **MCC** |
| Ant1.6 | 0.924 | 0.626 | 0.814 | 0.571 |
| Ant1.7 | 0.747 | 0.466 | 0.707 | 0.405 |
| jEdit4.1 | 0.821 | 0.552 | 0.841 | 0.568 |
| Xalan2.5.0 | 0.756 | 0.519 | 0.737 | 0.464 |
| Xalan2.6.0 | 0.811 | 0.632 | 0.802 | 0.564 |
| Xerces1.2.0 | 0.873 | 0.629 | 0.714 | 0.337 |
| Xerces1.3.0 | 0.885 | 0.693 | 0.786 | 0.446 |
| Xerces1.4.4 | 0.961 | 0.917 | 0.885 | 0.768 |

- Statistical validation:

To verify that the observed differences in predictive performance is statistically meaningful rather than attributable to random variation, non-parametric statistical test is employed. As the experiments involved eight software project datasets, each model produced a set of performance scores (AUC and MCC) per dataset.

The Wilcoxon signed-rank test is used to determine whether statistically significant differences existed between the two proposed FL architectures—FLCNN with HPT and FLANN with HPT—across all datasets. The Wilcoxon signed-rank test is selected because it does not assume data normality and is appropriate for small sample sizes. Statistical significance is established at a p-value threshold of 0.05. The p- value obtained based on AUC value is 0.0391 and based on MCC score is 0.0156. Both p-values are less than 0.05, this indicates that the observed differences in performance between FLCNN and FLANN are statistically significant, meaning the performance gap is unlikely due to random chance, and the two architectures exhibit genuinely different predictive capabilities.

Figure 6.17a shows the comparison of prediction performance of FLCNN before incorporating HPT and after incorporating HPT in terms of AUC measure. Figure 6.17b shows the comparison of prediction performance of FLCNN before incorporating HPT and after incorporating HPT in terms of MCC measure. The answer to RQ2 is described based on figure 6.17(a) and figure 6.17(b).




(a) (b)

Figure 6.17: Comparative Analysis of FLCNN with HPT and without HPT in terms of AUC (a) and MCC (b)

- · *RQ2: How does HPT impact the prediction accuracy of FL models?*

To answer RQ2, we compared the results of the FLCNN after incorporating HPT with the model proposed in our previous study. The hyperparameters such as dropout rate, learning rate, epoch count, and batch size is adjusted using GWO to increase the prediction performance of FLCNN model. According to figure 6.17a and figure 6.17b, incorporation of HPT in FLCNN model improves the prediction performance of FLCNN model for SDP in terms of both AUC and MCC measures. Across all the datasets, there is a significantly increase in AUC and MCC measures, which shows the positive impact of incorporating HPT in FLCNN model for SDP.

Based on RQ1, we can analyze that FLCNN with HPT model outperforms FLANN with HPT model and based on RQ2, we can analyze that FLCNN with HPT model outperforms FLCNN without HPT model in terms of both AUC and MCC values. So, we build the flask api of FLCNN with HPT model and then we create the docker image of the created flask api in order to analyze the security vulnerabilities in the model. Note that we have utilized freely available vulnerability scanning tools such as anchore, aqua trivy, snyk, and jfrog xray. The answer to RQ3 is described based on table 6.17 and table 6.18.

- · *RQ3: What security vulnerabilities are present in the Dockerized FL models, and how can they be identified?*

To address the answer of RQ3, table 6.17 describes the results obtained for all the chosen tools. All the tools categorize the security vulnerabilities into four types based on severity levels: Critical, High, Medium, and Low. According to table 6.17, jFrog Xray only detects 6 vulnerabilities in NumPy, pip, and mpmath package of the python3 version. All the vulnerabilities identified in the NumPy package already got fixed in their enhanced versions. In the case of Aqua Trivy also, we got only one medium severity vulnerability in the npm package which got fixed in its enhanced version but all the remaining vulnerabilities are non-fixed. Snyk tool categorized the vulnerabilities into two parts: vulnerabilities in the base image (jupyter/scipy-notebook and ermaker/keras-jupyter) and vulnerabilities in the TensorFlow library. The result of the snyk tool shows that in the case of the base image, all 185 vulnerabilities show no known exploit issue and in the case of TensorFlow, 11 vulnerabilities show proof of concept issue and 290 vulnerabilities show no known exploit issue. We have also analyzed that in the case of anchore tool, there exist 886 total dependencies and it detects 819 vulnerabilities, in the case of snyk tool, there exist 503 total dependencies and it detects 486 known vulnerabilities. There exist a greater number of medium severity vulnerabilities in the case of almost all the chosen tools.

Table 6.17: Severity wise security vulnerabilities for each tool

| **Tool Name** | **Critical** | **High** | **Medium** | **Low** |
|---|---|---|---|---|
| Anchore | 1 | 1 | 625 | 192 |
| Aqua Trivy | 0 | 2 | 689 | 258 |
| jFrog Xray | 0 | 4 | 2 | 0 |
| Snyk | 4 | 91 | 217 | 174 |

The most common potential threats identified in various packages are out of-bound, buffer overflow, information exposure, denial of service, arbitrary code injection, memory leak, authorization bypass, access restriction bypass, prototype pollution, divide by zero, improper verification of the cryptographic signature, XML injection, etc. Table 6.18 summarizes potential threats with the severity of vulnerabilities identified in the TensorFlow framework.

Table 6.18: Threats with the severity level

| **Threat Name** | **Critical** | **High** | **Medium** | **Low** |
|---|---|---|---|---|
| Remote Code Execution | 0 | 4 | 0 | 0 |
| Uninitialized Memory Exposure | 0 | 1 | 0 | 0 |
| Heap Based Buffer Overflow | 0 | 7 | 2 | 6 |
| Buffer Overflow | 0 | 4 | 2 | 14 |
| Denial of Service | 1 | 14 | 55 | 64 |
| Arbitrary Code Execution | 1 | 0 | 0 | 0 |
| Numeric Truncation Error | 1 | 0 | 0 | 0 |
| NULL Pointer Dereference | 0 | 14 | 7 | 10 |
| Integer Overflow | 0 | 6 | 8 | 0 |
| Out of Bounds | 0 | 24 | 5 | 11 |
| Access of Uninitialized Pointer | 0 | 2 | 0 | 0 |
| Improper Input Validation | 0 | 8 | 8 | 4 |
| Use after Free | 0 | 2 | 0 | 0 |
| Use of Uninitialized Variable | 0 | 2 | 1 | 0 |
| Division by Zero | 0 | 0 | 2 | 0 |
| Race Condition | 0 | 1 | 0 | 0 |

Denial of Service threat has 134 vulnerabilities with CVE-IDs (CVE-2022- 29195, CVE-2022-21737, CVE-2022-21738, CVE-2021-41208, CVE-2020-15265, etc.). NULL Pointer Dereference has 31 vulnerabilities with CVE-IDs (CVE-2022-29207, CVE-2022-21739, CVE-2021-37666, CVE-2021-37671, CVE-2021-37667, CVE-2021-37638, etc.). Out of Bound threat has 24 vulnerabilities with CVE-IDs (CVE-2022-23574, CVE-2022-23560, CVE-2022-21730, CVE-2021-37639, CVE-2020-26271, etc.). Heap based buffer overflow has 15 vulnerabilities with CVE-IDs (CVE-2022-21740, CVE-2021-29578, CVE-2021- 29579, CVE-2020-15195). Remote code execution has 4 high severity vulnerabilities with CVE-IDs (CVE-2022- 29216, CVE-2021-41228). Improper input validation has 20 vulnerabilities with CVE-IDs (CVE-2022-21736, CVE-2021-37648, CVE-2021-37676, CVE-2021-37665, CVE-2021-37663). Arbitrary code execution, Numeric truncation error, and Denial of service threats have critical vulnerabilities with CVE-IDs (CVE-2021-37678, CVE-2020-15202, and CVE-2020- 15206,). Race condition and use of uninitialized variables has 1 high severe vulnerability with CVE-IDs (CVE-2022-23563 and CVE-2022-23573). Division by zero has 2 medium severe vulnerabilities with CVE-IDs (CVE-2022-21741, CVE-

2022-21725). Access of uninitialized pointer has 2 high severity vulnerabilities with CVE-IDs (CVE-2021-41214, CVE-2021-41219).

## 6.7 Comparison with Baseline Techniques

In this study, 8 distinct datasets from open-source repository are used to examine the prediction performance of FLCNN with HPT and FLANN with HPT model. Note that, we have used AUC and MCC measures to examine the prediction performance of proposed model. In this study, GWO is used to tune the hyperparameters of CNN and ANN algorithms. FLCNN with HPT model outperforms FLANN with HPT model. The efficacy of the proposed model is assessed in comparison to conventional ML methods, such as DT, KNN, LR, NB, and SVM, utilizing the identical dataset.

Table 6.19: Comparative analysis of proposed model with conventional approaches

| | **Ant1.6** | **Ant1.7** | **jEdit4.1** | **Xalan 2.5.0** | **Xalan 2.6.0** | **Xerces 1.2.0** | **Xerces 1.3.0** | **Xerces 1.4.4** |
|---|---|---|---|---|---|---|---|---|
| DT | 0.813 | 0.846 | 0.873 | 0.730 | 0.782 | 0.732 | 0.763 | 0.909 |
| KNN | 0.691 | 0.719 | 0.746 | 0.705 | 0.817 | 0.700 | 0.679 | 0.893 |
| LR | 0.725 | 0.726 | 0.612 | 0.731 | 0.833 | 0.604 | 0.604 | 0.853 |
| NB | 0.832 | **0.889** | **0.926** | 0.723 | **0.865** | 0.737 | 0.830 | 0.806 |
| SVM | 0.676 | 0.694 | 0.740 | 0.651 | 0.784 | 0.540 | 0.594 | 0.849 |
| FL with ANN | 0.696 | 0.775 | 0.873 | 0.733 | 0.761 | 0.838 | 0.769 | 0.883 |
| FL with CNN | 0.836 | 0.707 | 0.752 | 0.738 | 0.787 | 0.724 | 0.873 | 0.956 |
| FLANN with HPT | 0.814 | 0.707 | 0.841 | 0.737 | 0.802 | 0.714 | 0.786 | 0.885 |
| FLCNN with HPT | **0.924** | 0.747 | 0.821 | **0.756** | 0.811 | **0.873** | **0.885** | **0.961** |

The findings illustrated in table 6.19, indicate that the proposed FLCNN with HPT model surpasses the ML techniques in terms of AUC values while maintaining data privacy and security.

- Statistical validation

To ensure the reliability of the comparative results, non-parametric statistical validation is conducted to assess the significance of the performance differences across datasets. To compare overall performance differences among all competing approaches, including baseline ML models (NB, RF, SVM, KNN, and LR), FLCNN and FLANN architectures without HPT, and the proposed models, the Friedman test was employed. The Friedman test is a non-parametric statistical method that evaluates whether statistically significant performance differences exist among multiple models tested across several datasets, thereby determining if at least one model exhibits a notably different performance level.

Friedman test assigns rank to each models individually. The model with the highest mean value is considered as the best model and the model with the lowest mean value is considered as worst model. Table 6.20 represents the calculated mean ranks based on AUC scores. According to table 6.20, proposed model FLCNN_HPT is assigned highest rank for all the eight datasets. The test yielded a chi-square value of 28.1420 with a p-value of 0.0004, indicating significant differences in median ranks among the techniques (p-value less than 0.05).

Table 6.20: Mean rank of all models

| Models | Mean rank |
|---|---|
| FLCNN_HPT | 7.75 |
| NB | 6.5 |
| FL_CNN | 5.9375 |
| DT | 5.4375 |
| FL_ANN | 5.1875 |
| FLANN_HPT | 5.1875 |
| KNN | 3.750 |
| LR | 3.750 |
| SVM | 1.500 |

These result implies that at least one technique performs significantly different from the others, warranting post-hoc pairwise comparisons to determine specific model differences. As the performance of all the techniques are not equivalent, then we apply pairwise nemenyi post hoc test to assess the effectiveness of the proposed model. Nemenyi post hoc test returns the pairwise p-value between two models. Figure 6.18 presents

the results of the Nemenyi post-hoc test, which depicts pairwise p-values among all evaluated models. In the heatmap representation, lighter color shades correspond to lower p-values, indicating stronger evidence of statistically significant performance differences. The results show that the proposed FLCNN with HPT achieves significantly better performance than traditional ML models, particularly SVM, KNN, and LR at a significance level of $p < 0.05$. Conversely, the performance differences among FLCNN with HPT, FLANN with HPT, and FL with CNN were not statistically significant ($p > 0.05$), suggesting comparable performance levels among the FL models. Overall, the nemenyi post-hoc analysis substantiates the superior predictive capability of the proposed FLCNN with HPT model across the evaluated approaches.




Figure 6.18: Heatmap for nemenyi post hoc test based on AUC score

## 6.8 Conclusion

This chapter presented a comprehensive analysis of SDP models with dual emphasis on performance optimization and security assessment within containerized environments. The study explored four major experimental directions: (1) security vulnera-

bility analysis of official and verified Docker images, (2) optimization of DL models using hybrid metaheuristic approaches, (3) application of FL for decentralized defect prediction, and (4) integration of HPT to enhance predictive performance. The developed models were containerized using Docker, enabling modularity, scalability, and reproducibility across distributed computing environments.

A key contribution of this chapter lies in the rigorous evaluation of security vulnerabilities and resource utilization of Dockerized FL models. Multiple open-source container scanning tools, including Anchore, Aqua Trivy, JFrog Xray, Docker Scout, and Docker Scan were employed to assess potential vulnerabilities at both the image and package levels. Additionally, container performance was monitored using Prometheus, Grafana, cAdvisor, and Locust, providing granular insights into CPU, memory, disk I/O, and network usage during model execution. The experimental findings demonstrate that the proposed optimized and federated models achieved high predictive accuracy while maintaining computational efficiency and a secure deployment posture, making them suitable for industrial and cross-organizational software development environments where data privacy and security are paramount.

The insights gained from secure and optimized deployment of SDP models serve as a foundation for extending this research to another real-world domain, Human Activity Recognition (HAR). The next chapter introduces a Dockerized ensemble DL framework that leverages multiple architectures to enhance activity recognition accuracy and efficiency. Similar to the SDP framework, emphasis is placed on achieving a balance between performance optimization and deployment security, ensuring that the proposed system remains scalable and resource-efficient. The study evaluates the runtime performance and security vulnerabilities of the containerized HAR system, illustrating the versatility and generalizability of the proposed approach across different application domains.

# Chapter 7

# COMPREHENSIVE PERFORMANCE AND SECURITY ANALYSIS OF A DOCKERIZED ENSEMBLE DEEP LEARNING MODEL FOR HUMAN ACTIVITY RECOGNITION

## 7.1 Introduction

Human activity recognition (HAR) refers to the prediction of activities such as walking, sitting, or climbing stairs using data collected from sensors like accelerometers and gyroscopes. HAR is multidisciplinary in nature and has applications in healthcare, sports science, human–computer interaction, surveillance, gait analysis, and ambient assisted living [288][136][92]. Traditionally, video-based HAR approaches have

been explored, but they face limitations such as high cost, privacy concerns, and restricted coverage areas. In contrast, sensor-based approaches using smartphones, wearable devices, and inertial measurement units are cost-effective, scalable, and privacy-preserving, as they generate rich time-series data along multiple axes.

Early HAR systems relied on handcrafted features combined with classical ML models such as RF, SVM, and KNN. Although effective for simple activities, these methods struggle to capture complex temporal dependencies. With the advent of DL models such as CNNs and LSTMs have shown significant improvements by automatically extracting features from raw sensor data. CNNs specialize in local feature extraction, while LSTMs and BiLSTMs capture temporal dependencies. Hybrid architectures like CNN-LSTM and CNN-BiLSTM further improve recognition accuracy. More recently, attention-based methods (e.g., AttnHAR [253]), generative frameworks (e.g., MRFC-VAE [159]), and recurrent hybrids (e.g., CNN-GRU [132]) have set new benchmarks for HAR.

Despite advances in DL, three important challenges remain in the current HAR literature:

- Limited exploration of ensembles: Most studies employ a single DL model. Although ensembles can improve robustness and reduce variance, systematic exploration of stacking ensembles in HAR is limited.
- Lack of deployment-oriented evaluation: Prior work emphasizes accuracy but rarely examines resource efficiency (CPU, memory, network, disk) under deployment on mobile or edge devices.
- Neglect of security considerations: DL models deployed in containers can inherit vulnerabilities from base images or dependencies. However, HAR studies have not addressed security vulnerability analysis, even though it is essential in healthcare and assisted living applications.

To address these gaps, we propose DeepHAR, a Dockerized stacking ensemble framework for HAR. DeepHAR integrates CNN, LSTM, BiLSTM, CNN-LSTM, and CNN-

BiLSTM as base learners, with RF as meta-learner to combine their complementary strengths. Unlike prior work, our framework emphasizes not only predictive accuracy but also reproducibility, deployment efficiency, and container security.

The key contributions of this work are:

- Novel Stacking Ensemble for HAR: The first integration of five DL models in a stacking ensemble, achieving superior performance and robustness on three public datasets (WISDM, UCI-HAR, and FLAAP). Notably, this is the first work to apply DL-based ensembles on the FLAAP dataset, where prior studies relied only on RF [150].
- Reproducible Deployment Framework: A Dockerized HAR system with comprehensive resource utilization analysis using cAdvisor, Prometheus, and Grafana.
- Security Vulnerability Assessment: One of the first systematic evaluations of containerized HAR models using Anchore, Aqua Trivy, Docker Scout, and Snyk.
- Statistical Validation: Non-parametric Friedman and Nemenyi tests confirming DeepHAR's significant improvements over state-of-the-art HAR methods.

The remaining sections of this chapter are organized as follows: Section 2 presents the research methodology, including dataset description, techniques used, performance evaluation measures adopted to assess model effectiveness, tools used for security vulnerability analysis, and tools used for resource consumption analysis. Section 3 describes the results based on experimentation performed, statistical validation, security vulnerabilities identified, and utilization of resources in terms of CPU, memory, disk i/o, and network bandwidth. Finally, section 4 describes the conclusion of this chapter.

## 7.2 Research Methodology

This section outlines the procedures and methodologies employed in the empirical study. It provides a structured overview of the steps followed to design, implement, and evaluate the predictive model developed in this chapter. Specifically, it discusses

Table 7.1: Summarization of Characteristics of Datasets

| Dataset Name | Sampling Rate | No. of Subjects | No. of Activities | Devices | Sensors |
|---|---|---|---|---|---|
| WISDM [151] | 20 Hz | 29 | 6 | Wearable Device | Accelerometer |
| UCI HAR [21] | 50 Hz | 30 | 6 | Smartphone | Accelerometer and Gyroscope |
| FLAAP [150] | 100 Hz | 8 | 10 | Smartphone | Accelerometer and Gyroscope |

about the dataset used in this chapter, DL techniques and ensemble techniques used, performance evaluation measure used, tools used for security analysis, and tools used to evaluate resources consumed by the model in terms of CPU, memory, disk i/o, and network bandwidth. Our goal is to ensure both technical rigor and reproducibility, addressing common limitations in prior HAR studies such as lack of HPT details, unclear dataset splits, and limited deployment analysis.

### 7.2.1 Dataset

We have used WISDM, UCI HAR, and FLAAP dataset in this study. Kwapisz *et al.* [151] introduced WISDM dataset. They collected the data related to six activities. For each activity, the accelerator readings were collected along x-axis, y-axis, and z-axis. Anguita *et al.* [21] introduced UCI HAR dataset. They collected the data related to six activities. For each activity, the accelerator and gyroscope readings were collected along x-axis, y-axis, and z-axis. Kumar and Suresh [150] introduced FLAAP dataset. They collected the data related to 10 activities. For each activity, the accelerator and gyroscope readings were collected along x-axis, y-axis, and z-axis. Table 7.1 summarizes the characteristics of datasets used in this study. For WISDM and FLAAP, data was randomly split into 70% training and 30% testing. For UCI-HAR, we followed the standard split (7,352 training / 2,947 testing). A fixed random seed (42) was used for all splits.

### 7.2.2 Techniques

This paper introduces DeepHAR, a Dockerized stacking ensemble DL framework for HAR using sensor data. Unlike prior approaches that employ single DL architectures, DeepHAR integrates CNN, LSTM, BiLSTM, CNN-LSTM, and CNN-BiLSTM as base learners and employs a RF meta-learner to enhance classification robustness.

- CNN

CNN is a deep ANN that consists of many layers of neural networks. Each neural network layer consists of neurons. Convolutional layer, batch normalization layer, pooling layer, dropout layer, and dense layer plays a very important role in building CNN model [63]. The detailed description is given in section 2.6 of chapter 2.

The architecture view of the CNN model used in this study comprises of three levels of convolution each with two 2DCNN layers and filters of different kernel sizes in each layer, max pooling layer with a pool size of two, dropout layer with a dropout rate of 0.01 to minimize the likelihood of overfitting.

- LSTM

LSTM has input, output and forget gate. LSTM stores the required features and removes unwanted features with the help of forget gate [80]. To make predictions at next level, it uses its previous layer output as input. The detailed description is given in section 2.6 of chapter 2.

The architecture view of the LSTM model used in this study comprises of two layers of LSTM. Each layer consists of 128 hidden neurons and a dropout layer with a dropout rate of 0.2 is appended after each layer to mitigate overfitting. Batch normalization layer is also included to normalize the output of its preceding layer.

- BiLSTM

BiLSTM consists of two LSTMs, one is used to take the input in forward direction

(past to future) and other is used to take the input in backward direction (future to past) [100]. The detailed description is given in section 2.6 of chapter 2.

The architecture view of the BiLSTM model used in this study comprises of two layers of BiLSTM. Each layer consists of 128 hidden neurons and a dropout layer with a dropout rate of 0.2 is appended after each layer to mitigate overfitting. Batch normalization layer is also included to normalize the output of its preceding layer.

- CNNLSTM

In this architecture, convolutional layer extract features from input data and LSTM layer is used for sequence prediction. The input data is first processed through convolutional layer and then output of convolutional layer is fed to LSTM layer [216]. The detailed description is given in section 2.6 of chapter 2.

The architecture view of the CNNLSTM model used in this study comprises of two levels of time distributed convolution each with two 1DCNN layers and filters of different kernel sizes in each layer. The output of time distributed convolutional layer is fed to two layers of LSTM. Time distributed 1D Max pooling layer with a pool size of two is also used. Dropout layer is also used with a dropout rate of 0.01 to minimize the likelihood of overfitting. Batch normalization layer is also included to normalize the output of its preceding layer.

- CNNBiLSTM

In this architecture, convolutional layer is used to extract local features from input data and BiLSTM layer is used to handle long term dependencies. Unlike LSTM that uses past data, BiLSTM uses past and future data for prediction [39]. The input data is first processed through convolutional layer and then output of convolutional layer is fed to BiLSTM layer. The detailed description is given in section 2.6 of chapter 2.

The architecture view of the CNNBiLSTM model used in this study comprises of two levels of time distributed convolution each with two 1DCNN layers and filters of different kernel sizes in each layer. The output of time distributed convolutional layer

Table 7.2: Hyperparameter Details

| **Components of the Proposed Framework** | **Hyperparameters** |
|---|---|
| Verbose | 1 |
| Epochs | 50 |
| Activation Function | ReLu, Sigmoid (At last level) (CNN)<br>Tanh, Sigmoid (At last level) (LSTM)<br>Tanh, Sigmoid (At last level) (BiLSTM)<br>Relu, Tanh, Sigmoid (At last level) (CNN+LSTM)<br>Relu, Tanh, Sigmoid (At last level) (CNN+BiLSTM) |
| Optimizer | Adam |
| Loss function | Categorical cross entropy |
| Number of layers | 6 2DCNN<br>2 LSTM<br>2 BiLSTM<br>4 1DCNN+2 LSTM<br>4 1DCNN+2 BiLSTM |
| Regularization Method | Dropout |

is fed to two layers of BiLSTM. Time distributed 1D Max pooling layer with a pool size of two is also used. Dropout layer is also used with a dropout rate of 0.01 to minimize the likelihood of overfitting. Batch normalization layer is also included to normalize the output of its preceding layer.

These models were selected to ensure complementarity. CNN excels at local feature extraction, while LSTM/BiLSTM capture sequential dependencies. Their integration through stacking is expected to mitigate individual weaknesses. Table 7.2 depicts the hyperparameter details of DL models used for predicting the human activities.

The predicted values of these five models are combined to generate input data for the stacking ensemble approach. In the stacking approach, the output of the base model is used as input data for the meta learner, and the result is based on the prediction of the meta learner. In this study, we used RF classifier as meta learner because of its robustness to correlated features and its proven success in HAR tasks. The intuition

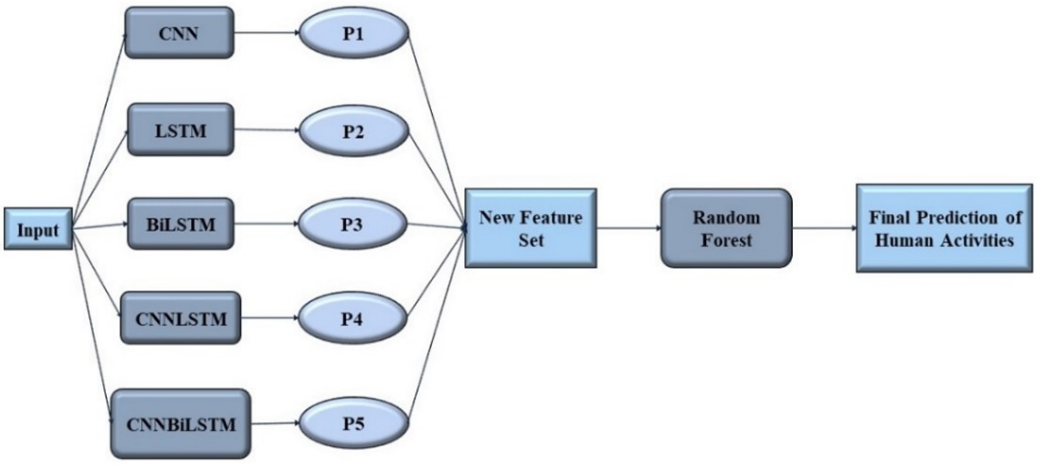



Figure 7.1: Architectural View of Our Proposed DeepHAR Model

is that different base models capture complementary error patterns: misclassifications from one model may be corrected by others. By learning these complementarities, the meta-learner improves overall accuracy and robustness. The systematic architecture of stacking is presented in figure 7.1.

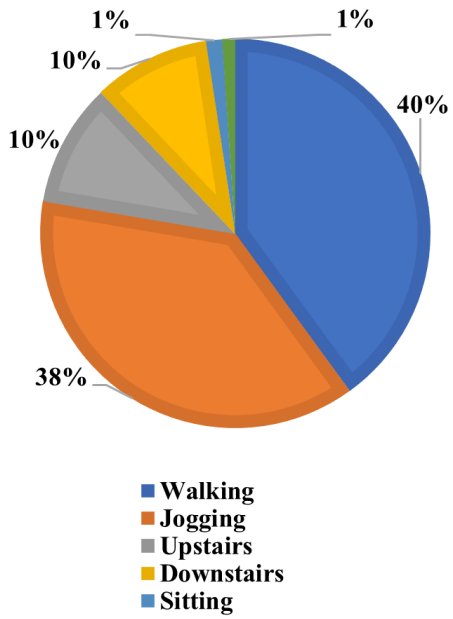



Figure 7.2: Activities in WISDM

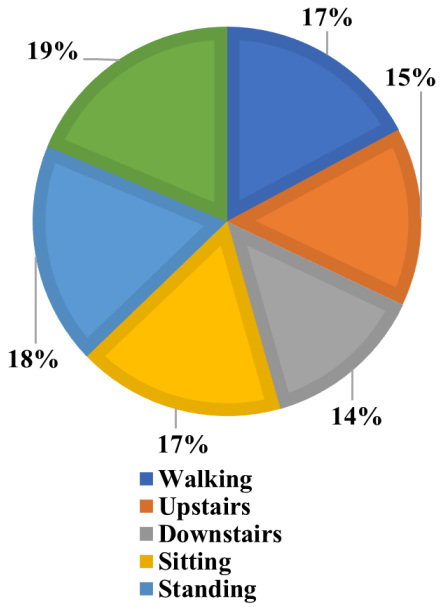



Figure 7.3: Activities in UCI HAR

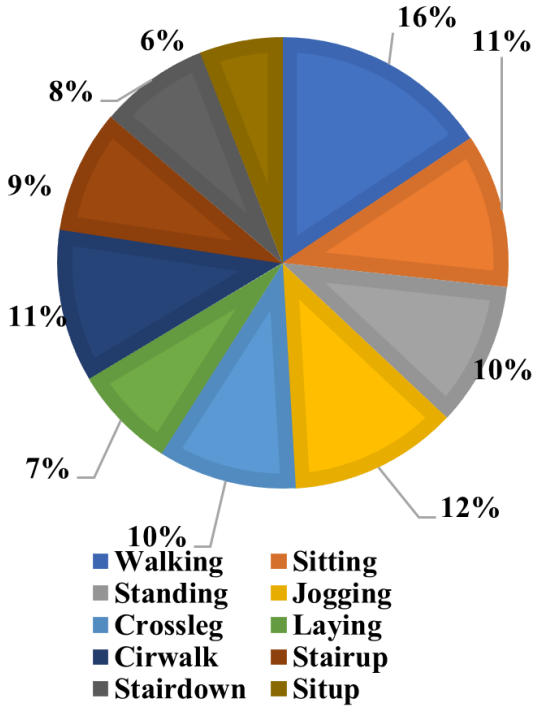



Figure 7.4: Activities in FLAAP

### 7.2.3 Performance Evaluation Measure

WISDM, UCI-HAR, and FLAAP datasets are imbalanced dataset as shown in figure 7.2, figure 7.3, and figure 7.4. The model can achieve high accuracy, if the classification technique predicts all the instances of dataset as a majority class. Therefore, accuracy is not considered as an appropriate measure for imbalanced dataset. AUC and MCC considers all the four quadrants of CM, making it well suited for situations where one class is much rarer than the other class. Thus, AUC and MCC is more useful measure to assess the performance of the model in case of imbalanced dataset.

AUC value represents the summary of the ROC curve. It depicts the relationship between FPR and TPR. The range of AUC value is 0 to 1. The correlation between actual and predicted data is represented by MCC value. The range of MCC is -1 to +1. We can determine the values of FPR, TPR, FP, TP, FN, and TN using CM. The detailed description of AUC and MCC is given in section 2.11 of chapter 2.

### 7.2.4 Tools for Security Vulnerability Analysis

Security issues may arise in ML and DL models since some packages may be outdated or contain some security vulnerabilities. A set of criteria was followed in the selection of container vulnerability scanning tools. We select commonly-used, open-source tools that have been already used by researchers and practitioners. The trial versions of paid product have limited functionalities, due to which free tools are taken into consideration. Here, security vulnerabilities in the docker image which contains all the files, packages, and libraries required to construct the ensemble of DL model for HAR are detected using Docker Scout, Anchore, Aqua Trivy, and Snyk [190]. We have installed tools extensions used in our study on the docker desktop and after that the generated docker image is analyzed to detect any security vulnerabilities. The detailed description of container vulnerability scanning tools is given in section 2.14 of chapter 2.

### 7.2.5 Tools for Resource Consumption Analysis

A set of criteria was followed in the selection of container performance monitoring tools. We select commonly-used, open-source tools that have been already used by researchers and practitioners. The trial versions of paid product have limited functionalities, due to which free tools are taken into consideration. This study makes use of open-source container performance monitoring tools, including cAdvisor, docker stats, Grafana, and Prometheus to assess the resource consumption. Prometheus is setup to collect relevant metrics that are made available by both docker and the application itself. cAdvisor is deployed as a container runtime monitor, gathering information about low-level resources utilization. Grafana is utilized to provide real-time performance insights through dashboards and visualizations. Figure 7.5 depicts the docker container monitoring architecture using cAdvisor, Prometheus, and Grafana. The data gathered from monitoring tools offers useful insights into the performance characteristics and resource requirements of the containerized environment during our experiments.

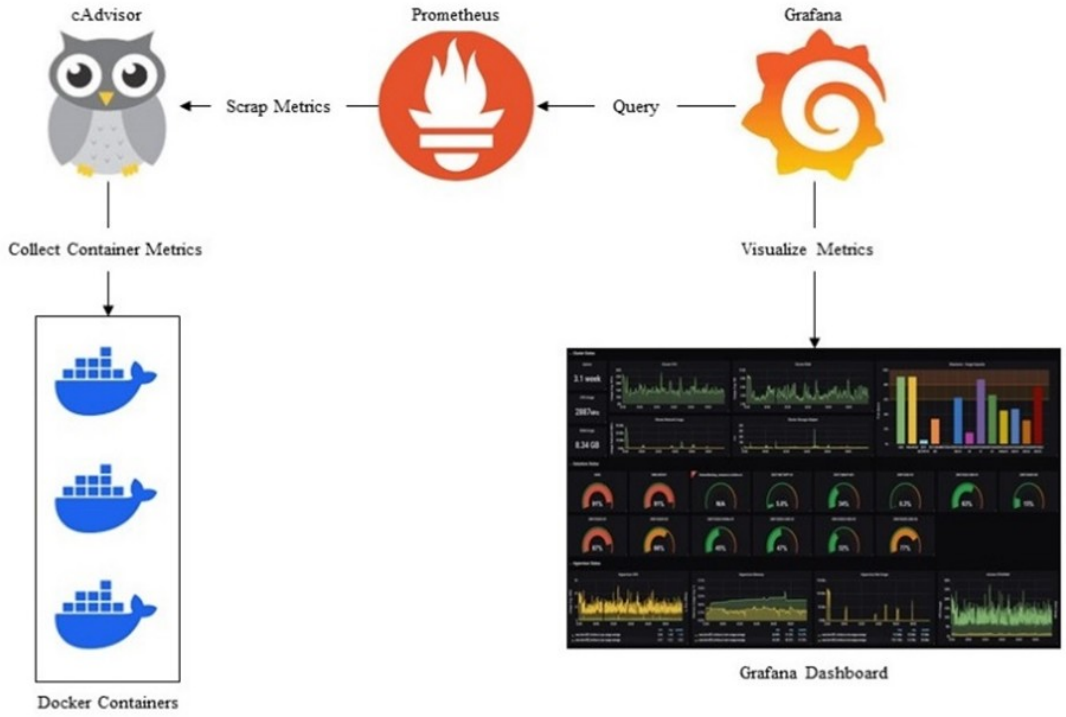



Figure 7.5: Docker Container Monitoring Architecture

### 7.2.6 Hardware Configuration Details

In this study, Google Colab is used to train and assess the performance of DL models. Tensorflow and Keras frameworks are used to generate the models. The server was running on a machine having x64 based processor and 192 GB RAM. Python, tensorflow, keras, and scikit-learn libraries are used for developing the models. All trained models were encapsulated in Docker images for reproducibility and portability.

## 7.3 Experimental Results

The experimental results of this study are discussed in this section. Firstly, the proposed model is evaluated on WISDM, UCI HAR, and FLAAP datasets. The results of this study are analyzed based on performance evaluation metrics. Table 4 depicts overall AUC and MCC values for WISDM, UCI-HAR, and FLAAP datasets. Accord-

Table 7.3: AUC and MCC values for WISDM, UCI-HAR, and FLAAP

| Dataset | Performance Evaluation Measure | CNN | LSTM | BiLSTM | CNN LSTM | CNN BiLSTM | DeepHAR | RF |
|---|---|---|---|---|---|---|---|---|
| WISDM | AUC | 0.975 | 0.978 | 0.975 | 0.965 | 0.980 | **0.992** | 0.815 |
| | MCC | 0.937 | 0.933 | 0.921 | 0.937 | 0.941 | **0.988** | 0.705 |
| UCI-HAR | AUC | 0.905 | 0.916 | 0.930 | 0.937 | 0.956 | **0.961** | 0.915 |
| | MCC | 0.770 | 0.866 | 0.814 | 0.887 | 0.896 | **0.921** | 0.798 |
| FLAAP | AUC | 0.868 | 0.878 | 0.850 | 0.870 | 0.869 | **0.935** | 0.669 |
| | MCC | 0.629 | 0.659 | 0.616 | 0.651 | 0.641 | **0.904** | 0.504 |

ing to table 4, among all the selected individual DL models, CNNBiLSTM performs better for all three considered datasets. Among the individual DL base models, RF and our proposed DeepHAR model, DeepHAR performs better for all three considered datasets with respect to AUC and MCC values.

### 7.3.1 Analysis on WISDM Dataset

WISDM dataset samples were partitioned into training dataset, which accounted for 70% of the sample, and a testing dataset, which accounted for remaining 30%. Training dataset is used to train the above-mentioned base DL models, while the test dataset is used to test the predictive effectiveness of proposed DeepHAR model. Figure 6(a) and figure 6(b) demonstrates the graphical representation of comparative analysis of five individual base models, RF, and our proposed DeepHAR model for each activity class with respect to AUC values and MCC values on WISDM dataset. it can be observed that the proposed ensemble model correctly classifies four activity classes (almost 100% accuracy), except upstairs and downstairs activity classes. It can also be observed that 4% of the samples in upstair activity class were misclassified as downstairs and 1.56% of the samples in downstairs activities were misclassified as upstairs, as these both activities are opposing kind of activities. Hence, it can be concluded that AUC and MCC values for each activity class with respect to other activity classes is almost equal to 1 for our proposed model, which indicates that our proposed DeepHAR model predicts all the activity classes efficiently.

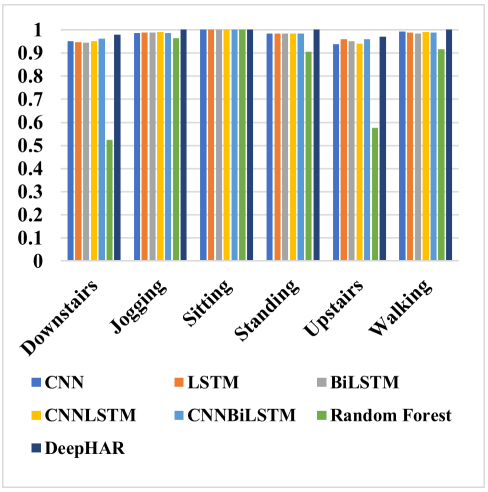



Fig (a) Activity wise AUC values

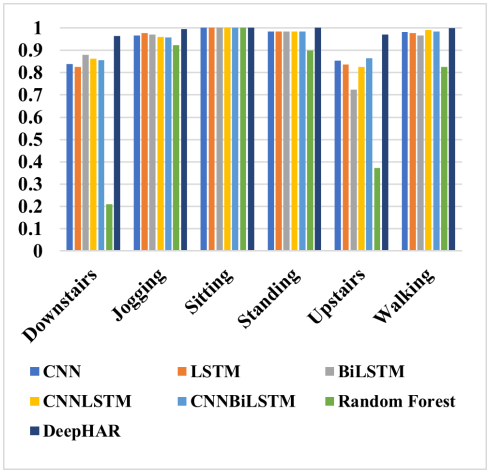



Fig (b) Activity wise MCC values

Figure 7.6: Activity wise AUC and MCC values for WISDM dataset

### 7.3.2 Analysis on UCI HAR Dataset

UCI HAR dataset consists of 10299 total samples, out of which 7352 samples were used as training data and 2947 samples were used as testing data. Training dataset is used to train the above-mentioned base DL models, while the test dataset is used to test the predictive effectiveness of proposed DeepHAR model. Figure 7(a) and figure 7(b) demonstrates the graphical representation of comparative analysis of five individual base models, RF, and our proposed DeepHAR model for each activity class with respect to AUC values and MCC values on UCI HAR dataset. it can be observed that except for the sitting and standing activity classes; all four other activity classes were correctly classified (almost 100% accuracy) by the proposed ensemble model. It can also be observed that 19.96% of the samples in sitting activity class were misclassified as standing activity and 8.65% of the samples in standing activity class were misclassified as sitting activity, as these both activities have similar characteristics in linear acceleration.

In addition to the above, 5.24% of the samples in walking activity class were misclassified as walking_downstairs activity and 4.03% of the samples in walking_upstairs activity class were misclassified as walking_downstairs activity. Hence, it can be concluded that the AUC and MCC values for four out of six activity classes are almost

equal to 1 for our proposed model. This indicates that our proposed DeepHAR model efficiently predicts all activity classes, except for two activity classes: standing and sitting.

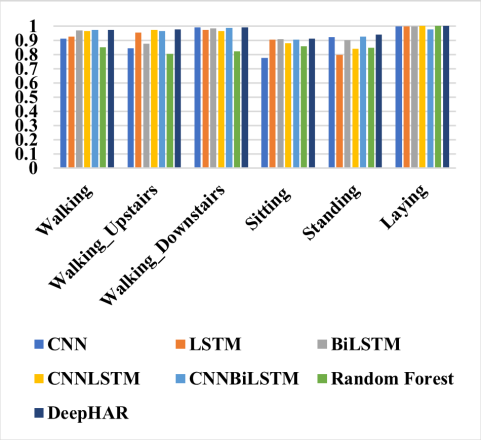



Fig (a) Activity wise AUC values

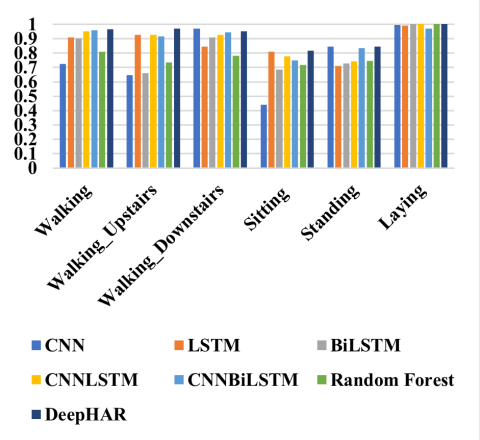



Fig (b) Activity wise MCC values

Figure 7.7: Activity wise AUC and MCC values for UCI HAR dataset

### 7.3.3 Analysis on FLAAP Dataset

FLAAP dataset samples were partitioned into training dataset, which accounted for 70% of the sample, and a testing dataset, which accounted for remaining 30%. Training dataset is used to train the above-mentioned base DL models, while the test dataset is used to test the predictive effectiveness of proposed DeepHAR model. Figure 8(a) and figure 8(b) demonstrates the graphical representation of comparative analysis of five individual base models, RF, and our proposed DeepHAR model for each activity class with respect to AUC values and MCC values on FLAAP dataset. it can be analyzed that 98.23% of the samples in walking class, 77.89% of the samples in jogging activity class, 86.94% of the samples in sitting activity class, 78.83% of the samples in cirwalk activity class, 92.79% of the samples in standing activity class, 89.67% of the samples in crossleg activity class, 86.49% of the samples in stairup activity class, 93.51% of the samples in stairdown activity class, 80.93% of the samples in laying activity class, and 95.85% of the samples in situp activity class were correctly classified by the proposed ensemble model.

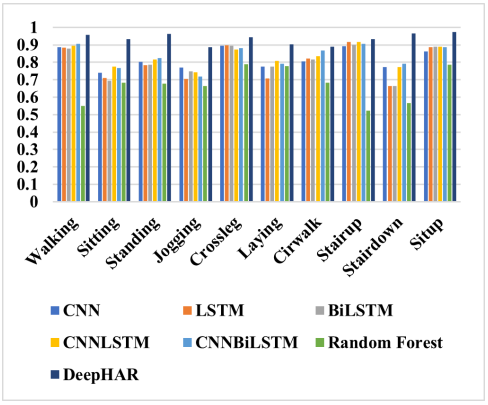



Fig (a) Activity wise AUC values

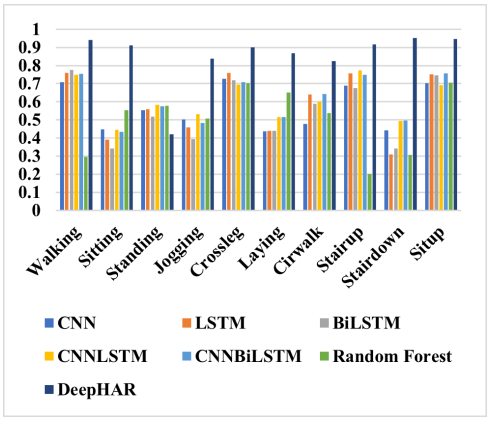



Fig (b) Activity wise MCC values

Figure 7.8: Activity wise AUC and MCC values for FLAAP dataset

In addition to the above, it can also be observed that 11.56% of the samples in jogging activity class were misclassified as walking activity and 5.53% of the samples in jogging activity class were misclassified as cirwalk activity, 16.06% of the samples in cirwalk activity class were misclassified as walking activity, and 13.95% of the samples in laying activity class were misclassified as walking activity.

It can be concluded that AUC and MCC values for each activity class with respect to other activity classes is almost greater than 0.9 for our proposed model, which indicates that our proposed DeepHAR model predicts all the activity classes efficiently.

From table 4, it can be inferred that proposed DeepHAR model exhibits higher AUC and MCC values as compared to base DL models and RF. Furthermore, a statistical test is employed to validate the accuracy of proposed DeepHAR model's predictions. Subsequently, the proposed model is deployed on docker to evaluate its performance in terms of resource consumption and assess its security vulnerabilities.

### 7.3.4 Statistical Test

In the previous sections, we have discussed the experimental results of the proposed model on WISDM, UCI HAR, and FLAAP datasets and we found that the proposed DeepHAR model outperforms base DL models in terms of AUC and MCC. Furthermore, we have also used non parametric statistical test called Friedman test to validate

the effectiveness of DeepHAR model over the base DL models and RF based on AUC and MCC metrics.

Friedman test is a non parametric statistical test which is used to evaluate the statistically significant difference between the models, compare the performance of the models, and evaluate the rankings of the models. Friedman test assigns ranks to each models individually. The model with the highest mean value is considered as the best model and the model with the lowest mean value is considered as worst model. Table 5 and table 6 represents the calculated mean ranks based on AUC and MCC metrics. According to table 5 and table 6, proposed model is assigned highest rank for all the three datasets.

Table 7.4: Mean Ranks based on AUC values

| **Techniques** | **Mean Rank** |
|---|---|
| CNN | 2.50 |
| LSTM | 4.67 |
| BiLSTM | 3.17 |
| CNNLSTM | 4.00 |
| CNNBiLSTM | 5.33 |
| Random Forest | 1.33 |
| Proposed Model (DeepHAR) | 7.00 |

Table 7.5: Mean Ranks based on MCC Values

| **Techniques** | **Mean Rank** |
|---|---|
| CNN | 2.83 |
| LSTM | 4.33 |
| BiLSTM | 2.33 |
| CNNLSTM | 4.83 |
| CNNBiLSTM | 5.33 |
| Random Forest | 1.33 |
| Proposed Model (DeepHAR) | 7.00 |

The null hypothesis states that there is no statistically significant difference between the performance of the techniques. To justify the null hypothesis, we compare the calculated value of friedman statistics with the chi square tabulated value at degree of freedom (df) k-1, where k represents number of techniques.

The calculated value of the friedman statistics based on table 8 is 13.760 and table 9 is

14.766. It can be observed from chi square table, that at k-1 df, the friedman statistic value at 0.05 significant is 12.592 which is less than calculated value for both table 8 and table 9. Therefore, we reject the null hypothesis. Hence, we may deduce that there exists statistically significant difference between the performance of the proposed model and other models. As the performance of all the techniques are not equivalent, then we apply pairwise nemenyi post hoc test to assess the effectiveness of the proposed model. Nemenyi post hoc test returns the pairwise p-value between two models.

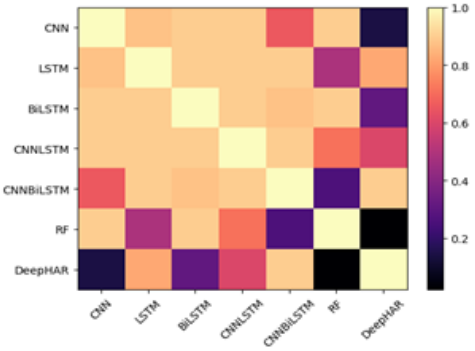



Fig (a) Pairwise nemenyi post hoc scores based on AUC values

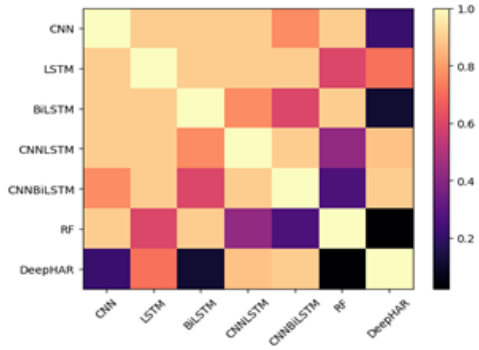



Fig (b) Pairwise nemenyi post hoc scores based on MCC values

Figure 7.9: Heat map of pairwise nemenyi post hoc scores based on AUC and MCC values

Figure 9(a) shows the heatmap for pairwise nemenyi post hoc analysis of proposed model and other models for based on AUC values. According to figure, there exist a statistically significant difference between the performance of proposed DeepHAR model and other base DL models and RF model. Figure 9(b) shows the heatmap for pairwise nemenyi post hoc analysis of proposed model and other models based on MCC values. According to figure, there exist a significant difference between the performance of proposed model and other models.

### 7.3.5 Security Analysis

This study utilizes open-source container vulnerability scanning tools, including docker scout, aqua trivy, anchore, and snyk to assess the security vulnerabilities in docker container. The security vulnerabilities are categorized into four levels: Critical, High,

Medium, and Low, according to their severity levels. The security vulnerabilities identified using each tool is shows in table. Anchore recognizes 357 vulnerabilities, Aqua Trivy detects 334 vulnerabilities, Docker Scout detects 150 vulnerabilities, and Snyk detects 65 vulnerabilities.

Table 7 demonstrates that the docker image of the proposed DeepHAR model is secure because a smaller number of critical and high-severity vulnerabilities is identified for each of the selected tools.

Table 7.6: Security Vulnerabilities

| **Tools** | **Critical** | **High** | **Medium** | **Low** |
|---|---|---|---|---|
| Anchore | 0 | 15 | 187 | 155 |
| Aqua Trivy | 0 | 8 | 177 | 149 |
| Docker Scout | 0 | 3 | 87 | 60 |
| Snyk | 1 | 1 | 32 | 31 |

We detected some vulnerabilities that are common to all the selected tools and occur frequently during the scanning of docker images. Table 8 depict the data regarding common vulnerabilities that arise frequently in the selected tools. These security vulnerabilities increase the possibility of high resource consumption, denial-of-service attack, loss of confidentiality, leakage of sensitive information, etc.

Table 7.7: CVEs Identified

| **S.no.** | **CVE-ID** | **Level of Severity** | **Description of CVEs** |
|---|---|---|---|
| 1 | CVE-2021-35958 | Critical | Arbitrary file writes via archive extraction |
| 2 | CVE-2022-42919 | High | Improper privilege management |
| 3 | CVE-2022-29217 | High | Use of a broken or risky cryptographic algorithm |
| 4 | CVE-2023-0286 | High | Access of resources using incompatible types which leads to denial of service |
| 5 | CVE-2023-38427 | Medium | Out of bound read or integer underflow |

| | | | |
|---|---|---|---|
| 6 | CVE-2022-45061 | Medium | Inefficient Algorithmic Complexity which leads to CPU denial of service |
| 7 | CVE-2023-24329 | Medium | Improper input validation which leads to compromised integrity |
| 8 | CVE-2021-21240 | Low | Uncontrolled resource consumption which leads to system unavailability |
| 9 | CVE-2023-23931 | Low | Improper check for unusual or exceptional conditions which leads to corrupted output. |
| 10 | CVE-2023-6237 | Low | Excessive time spent checking invalid RSA public keys |

### 7.3.6 Performance Analysis

It should be noted that this study utilized open-source container monitoring tools, including docker stats, cAdvisor, Grafana, and Prometheus to monitor the resource consumption of docker container. An extensive visualization and monitoring setup is created by combining cAdvisor, Grafana, and Prometheus. cAdvisor (Container Advisor), an open-source tool, is used for monitoring container resources. It collects and assesses metrics related to resource consumption of containerized applications. cAdvisor is incorporated into docker desktop to monitor the utilization of CPU, allocated memory, network traffic consumption, and disk utilization by the containers that have neem deployed. We utilize Prometheus to gather and analyze relevant resource usage metrics from our experimental environment. Prometheus consistently retrieves resource consumption data from cAdvisor and stores it. We utilized Grafana, a freely available data visualization tool, to visually represent the data that was stored by Prometheus.

Resource consumption metrics such as utilization of CPU, allocated memory, network traffic consumption, and disk utilization were gathered at the interval of 10 seconds during a monitoring period spanning 6 hours. Table 9 shows the resource consumption (Memory allocated, CPU usage, disk usage, and network traffic usage) of proposed ensemble model DeepHAR using monitoring tools. According to table 9, Grafana shows high resource utilization with CPU utilization 13.97%, 1.70 TB memory allo-

cated, network traffic at 150 KB received and 0B sent, and disk usage at 150 MB over a monitoring period of 6 hours among all the selected monitoring tools.

Table 7.8: Resource Consumption using Container Monitoring Tools

| **Tools** | **Utilization of CPU** | **Allocated Memory** | **Network Traffic Consumption** | **Disk Usage** |
|---|---|---|---|---|
| cAdvisor | 12 | 1.95 GB | 1.45 KB/0B | 108 KB |
| Docker Stats | 76 | 1.25 GB | 1.5 KB/0 B | 0B |
| Prometheus | 13 | 1.586 GB | 150 KB/0B | 150 MB |
| Grafana | 13.97 | 1.70 TB | 150 KB/0B | 150 MB |

Figure 10 shows the CPU usage trend, figure 11 shows the memory allocation trend, and figure 12 shows the network traffic data received using Prometheus. In figure 10, 11, and 12, x- axis indicates time frame. In figure 10, y-axis indicates CPU utilization in percentage, in figure 11, y-axis indicates memory allocated in kilo bytes, and in figure 12, y-axis indicates network traffic received in kilo bytes.

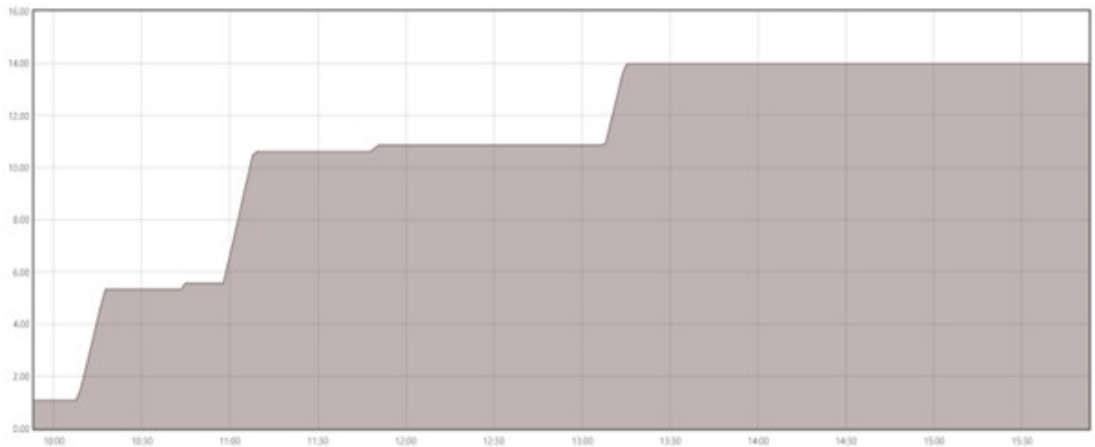


Figure 7.10: Prometheus CPU usage

The result shows that over a monitoring period of 6 hours, CPU usage is 13%, memory allocated is 1.586 GB, network traffic received is 150 KB and network traffic sent is 0B, and total bytes used are approximately 150 MB.

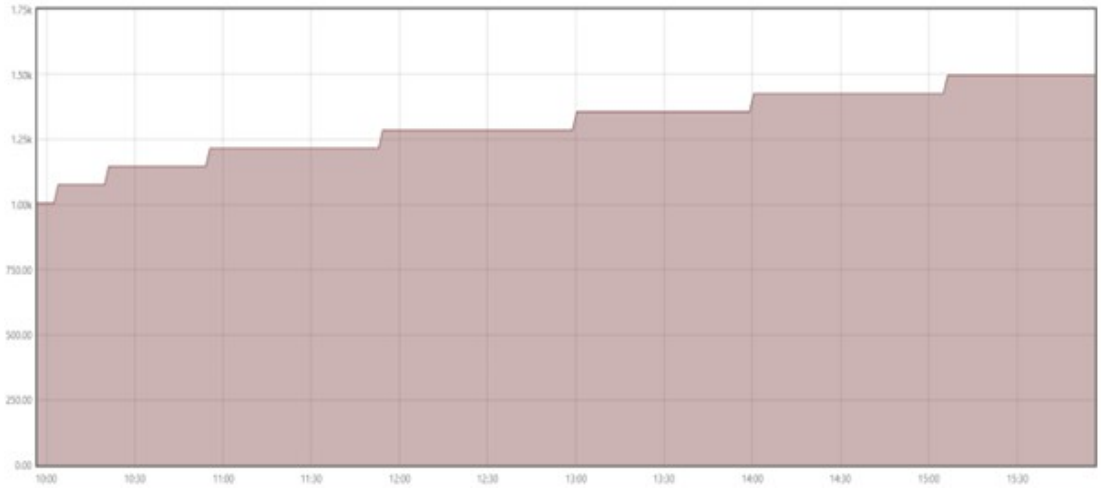


Figure 7.11: Prometheus memory usage

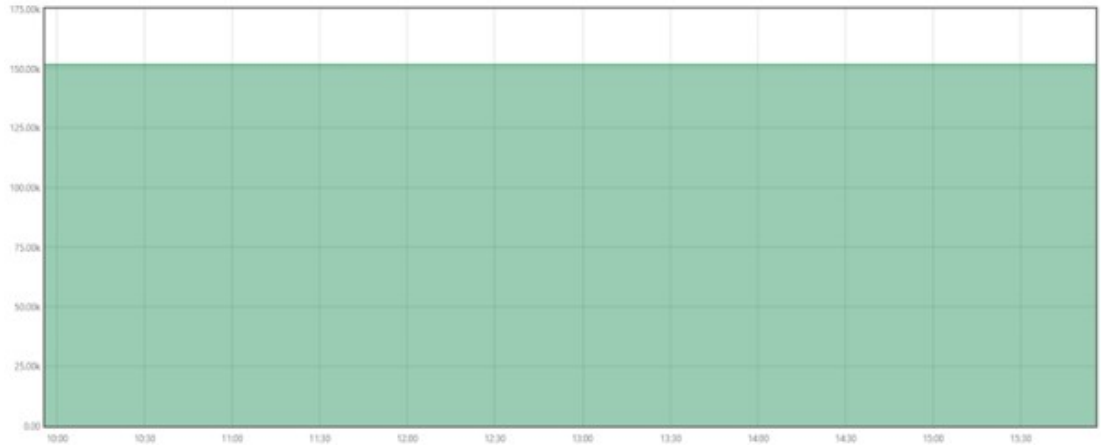


Figure 7.12: Prometheus network traffic received

Figure 13 shows the CPU usage trend, figure 14 shows the memory allocation trend, and figure 15 shows the disk storage in using Grafana dashboard. In figure 13, 14, and 15, x- axis indicates time frame. In figure 13, y-axis indicates CPU usage in percentage, in figure 14, y-axis indicates memory allocated in kilo bytes, and in figure 15, y-axis indicates disk storage in megabytes. The result shows that over a monitoring period of 6 hours, CPU usage is approximately 14%, memory allocated is 1.70 TB, network traffic received data is 150 KB and sent data is 0B, and total bytes used are approximately 150 MB.

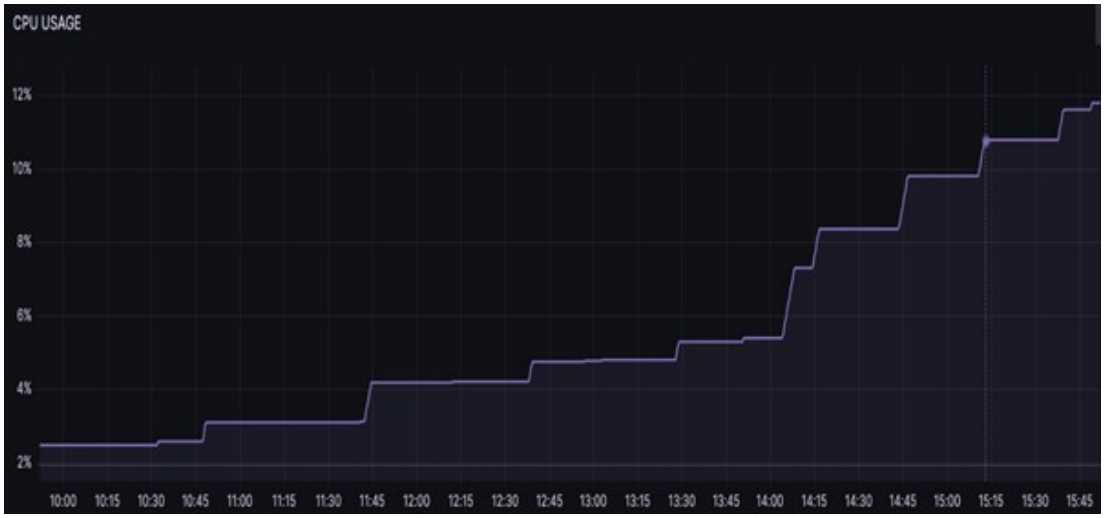



Figure 7.13: Grafana CPU usage




Figure 7.14: Grafana memory usage

## 7.4 Conclusion

This chapter presented a comprehensive evaluation of DeepHAR, a Dockerized stacking ensemble framework that integrates multiple deep learning architectures—CNN, LSTM, BiLSTM, CNN-LSTM, and CNN-BiLSTM—with a RF meta-learner for HAR. Experimental results on the WISDM, UCI-HAR, and FLAAP datasets showed that DeepHAR consistently outperforms individual deep models and classical classifiers in both AUC and MCC, with Friedman and Nemenyi statistical tests confirming the

significance of these improvements.

Crucially, the chapter extended HAR research beyond predictive accuracy to address deployment realities such as resource efficiency and container security. Security analysis with Anchore, Aqua Trivy, Docker Scout, and Snyk revealed a low number of critical or high-severity vulnerabilities, supporting the robustness of the Dockerized deployment in practical contexts. Performance monitoring using cAdvisor, Prometheus, and Grafana demonstrated that DeepHAR efficiently utilizes CPU, memory, disk, and network resources, making it suitable for real-world and resource-constrained settings.

Overall, DeepHAR establishes a new benchmark for robust, scalable, and secure human activity recognition—balancing high accuracy with practical deployment considerations. This work addresses key limitations of earlier HAR literature and paves the way toward reproducible, privacy-preserving, and trustworthy AI systems for human-centric applications.

# Chapter 8

# CONCLUSION

## 8.1 Summary of the Research Work

With the rapid adoption of cloud services, containerization, and DL technologies, organizations faces critical challenges related to unreliable resource allocation, security vulnerability in containerized environments, inefficient fault management, and lack of privacy-preserving intelligence. This thesis contributes to overcoming these challenges by proposing novel FL based frameworks, metaheuristic-driven optimization techniques, and Dockerized microservice deployments across multiple domains of software and data-driven systems. The research presented in this thesis aims to address several emerging challenges in software engineering, CC, and artificial intelligence, particularly focusing on software quality assurance, workload forecasting, failure prediction, and secure model deployment.

This thesis presented a comprehensive exploration of ML, DL, and FL paradigms for solving critical challenges in CC, software engineering, and intelligent systems. The overarching goal was to design and evaluate scalable, secure, and efficient predictive frameworks that leverage advanced data-driven techniques for enhancing automation, reliability, and resource utilization across distributed computing environments.

The research work presented in this thesis spans seven chapters, each addressing a

distinct yet interrelated objective. The initial chapters laid the conceptual foundation for end to end predictive modelling, FL, deep neural networks, container technologies, and tools used for security vulnerability and resource utilization analysis, while the subsequent experimental chapters (Chapters 3–7) provided empirical investigations and contributions. Together, these studies form a cohesive framework for developing secure, privacy-preserving, and resource-efficient DL models that are applicable in cloud, software engineering, and human-centered domains.

The research began with an extensive systematic literature review focuses on the maintenance, monitoring, and security of containers, which provides an in depth understanding of research gaps, existing challenges, and emerging trends in containerized software maintenance.

## 8.2 Application of the Work

The outcomes and contributions of this research have substantial implications across academia, software engineering practice, and society. The proposed frameworks provide practical solutions for real-world challenges such as software defect prediction, workload management, cloud failure prediction, and secure deployment, while also contributing to foundational advancements in federated and containerized learning.

### 8.2.1 Aid to Researchers

This research provides an academic foundation for future studies in federated optimization, secure deployment, and adaptive cloud intelligence.

- Benchmark Frameworks:

The experimental setups in workload forecasting, failure prediction, SBP, and HAR serve as reproducible reference architectures for subsequent research. Researchers can use the published datasets, preprocessing strategies, and hyperparameter tuning methodologies as comparative baselines.

- Advancement in Federated Learning Research:

The thesis introduces novel applications of FL in cloud workload and software quality domains, encouraging exploration of privacy-preserving intelligence in other domains such as healthcare, IoT, and smart cities.

- Interdisciplinary Integration:

The research bridges software engineering, artificial intelligence, and CC, paving the way for integrated studies on AIOps, predictive DevOps, and automated quality assurance.

- Open Research Opportunities:

The methodologies and findings highlight potential for extensions in federated transfer learning, explainable federated models, and blockchain-enhanced secure collaboration

### 8.2.2 Aid to Software Practitioners

The research provides actionable insights and technical frameworks for practitioners managing large-scale cloud systems and software development environments.

- Predictive Maintenance and Resource Optimization:

The proposed Federated CNN-based workload forecasting and failure prediction frameworks can be integrated into cloud orchestration systems to enhance auto-scaling, load balancing, and early anomaly detection.

- Automated Bug Detection:

The FL-based software defect prediction models enable proactive identification of defect-prone modules before deployment, improving reliability and reducing testing costs.

- Secure and Portable Model Deployment:

The Dockerized model architecture ensures modularity, reproducibility, and security compliance, supporting multi-cloud and hybrid deployment scenarios.

- Continuous Security Monitoring:

The inclusion of multiple container vulnerability scanning tools within the deployment pipeline demonstrates a practical implementation of DevSecOps, ensuring continuous risk mitigation.

### 8.2.3 Aid to industry

Industrial stakeholders can benefit from the proposed techniques in several ways:

- Cloud Service Optimization:

Service providers can integrate the FL-based workload prediction models for cost-effective resource provisioning, minimizing energy consumption and operational expenses.

- Reliable Software Delivery:

The defect prediction frameworks ensure stable software releases, thereby increasing customer satisfaction and reducing post-deployment maintenance.

- Healthcare and IoT Applications:

The Dockerized HAR framework can be utilized in remote health monitoring, fitness tracking, and ambient assisted living systems, ensuring accurate and privacy-preserving activity recognition.

### 8.2.4 Aid to Society

- Privacy and Ethics:

The FL approach ensures that sensitive user data remains local, aligning with privacy regulations like GDPR and promoting ethical AI.

- Public Welfare and Safety:

The proposed HAR system can be deployed in elderly care and surveillance applications, assisting in timely detection of abnormal behaviors or emergencies.

- Sustainability:

Efficient workload forecasting contributes to reduced energy consumption in cloud data centers, advancing green computing initiatives.

## 8.3 Future of the Work

While this research has made several novel contributions in advancing the state of FL, optimization, and deployment for building the intelligent and secure software systems, there remain open challenges and promising avenues for future exploration. Future research can extend, refine, and apply these frameworks to tackle emerging challenges in distributed AI, cloud-edge collaboration, and trustworthy automation.

- Energy-Efficient Federated Models:

While the research work in this thesis focuses on improving accuracy, security, and privacy, future work can explore energy-aware optimization during model training and deployment, particularly for edge and IoT environments where computational resources are constrained.

- Explainable and Interpretable FL:

FL model operates as black boxes, creating a gap in model interpretability and trustworthiness. Future research can focus on building explainable FL model. Developing explainable federated models will enhance transparency and trust in AI-driven decision-making, particularly in safety-critical domains.

- Integration with Blockchain:

Blockchain can be integrated with FL to ensure secure aggregation and verifiable training, eliminating risks of data tampering or malicious updates. This integration can be beneficial in solving problems like federated SDP or multi cloud workload management where participants belong to different administrative domains as the integration of blockchain strengthen the reliability and traceability of FL communications.

- Cross-Domain Transfer Learning:

The datasets used in this thesis belong to specific domains such as software engineering, workload management, and HAR. However, real-world systems often experience domain shifts and non-IID data distributions. Future work should explore cross-domain transfer learning and domain adaptation strategies. Extending the proposed models to handle cross-domain and heterogeneous data could improve generalization and adaptability in multi-tenant cloud environments.

- Advanced Container Security:

This thesis focuses on security analysis of dockerized models using static vulnerability scanning tools. Future research should focus on the implementation of dynamic, real-time anomaly detection, container runtime security, and self-healing orchestration mechanism. Integration of runtime security analysis tools into the predictive pipeline will enable continuous security assurance and resilient DevSecOps practices.

- Hybrid Cloud–Edge Deployment:

The datasets used in this thesis belong to specific domains such as software engineering, workload management, and human activity recognition. However, real-world systems often experience domain shifts and non-IID data distributions. Future work should explore cross-domain transfer learning and domain adaptation strategies. Deploying FL models in hybrid environments combining cloud and edge devices would reduce latency, improve scalability, and enhance privacy.

- Human-Centric AI Systems:

In the context of the Dockerized HAR framework, future research could extend beyond basic motion detection to include emotion-aware and context-sensitive recognition using multi-modal data sources such as physiological sensors, voice, and facial cues. Expanding the HAR framework to include emotion detection and behavioral analysis will enable the development of more adaptive, human centered AI systems, and personalized human–computer interaction systems.

Finally, we can conclude that this thesis presents a comprehensive, end-to-end research framework that bridges critical gaps between ML innovation and production-grade deployment. Specifically, it addresses three interconnected challenges in modern CC: (1) ensuring model accuracy under data heterogeneity and distribution constraints, (2) preserving privacy and security throughout the ML lifecycle, and (3) enabling trustworthy, cross-organizational collaboration without compromising operational efficiency.

Through eight empirically rigorous studies spanning workload forecasting, cloud failure prediction (job and task failures), software defect detection, and human activity recognition, this work demonstrates that high predictive accuracy, data privacy, resource efficiency, and container-level security are not competing objectives—but rather complementary goals achievable through systematic architectural design and rigorous validation. By expanding upon the models and methodologies developed in this thesis, future studies can establish the foundation for next-generation self-managing, privacy-

preserving, and sustainable intelligent systems that will power modern software and cloud ecosystems.

# Bibliography


[1] *1. fio - Flexible I/O tester rev. 3.27 — fio 3.27-169-g1953e1-dirty documentation*. URL: https://fio.readthedocs.io/en/latest/fio_doc.html.

[2] Omar Arif Abdul-Rahman and Kento Aida. "Towards understanding the usage behavior of Google cloud users: the mice and elephants phenomenon". In: *2014 IEEE 6th International Conference on Cloud Computing Technology and Science*. IEEE. 2014, pp. 272–277.

[3] Subil Abraham et al. "On the Use of Containers in High Performance Computing Environments". In: *2020 IEEE 13th International Conference on Cloud Computing (CLOUD)*. 2020, pp. 284–293. DOI: 10.1109/CLOUD49709.2020.00048.

[4] Amro Abuabdo and Ziad A Al-Sharif. "Virtualization vs. Containerization: Towards a Multithreaded Performance Evaluation Approach". In: *2019 IEEE/ACS 16th International Conference on Computer Systems and Applications (AICCSA)*. 2019, pp. 1–6. DOI: 10.1109/AICCSA47632.2019.9035233.

[5] Ashijeet Acharya et al. "A Performance Benchmarking Analysis of Hypervisors Containers and Unikernels on ARMv8 and x86 CPUs". In: *2018 European Conference on Networks and Communications (EuCNC)*. 2018, pp. 282–289. DOI: 10.1109/EuCNC.2018.8443248.

[6] Jigna N Acharya and Anil C Suthar. “Docker Container Orchestration Management: A Review”. In: *International Conference on Intelligent Vision and Computing*. Springer. 2022, pp. 140–153.

[7] Mainak Adhikari and Satish Narayana Srirama. “Multi-objective accelerated particle swarm optimization with a container-based scheduling for Internet-of-Things in cloud environment”. In: *Journal of Network and Computer Applications* 137 (2019), pp. 35–61. ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2019.04.003. URL: https://www.sciencedirect.com/science/article/pii/S1084804519301158.

[8] Imtiaz Ahmad et al. “Container scheduling techniques: A Survey and assessment”. In: *Journal of King Saud University - Computer and Information Sciences* (2021). ISSN: 1319-1578. DOI: https://doi.org/10.1016/j.jksuci.2021.03.002. URL: https://www.sciencedirect.com/science/article/pii/S1319157821000562.

[9] Md. Razu Ahmed et al. “The Impact of Software Fault Prediction in Real-World Application: An Automated Approach for Software Engineering”. In: *Proceedings of 2020 the 6th International Conference on Computing and Data Engineering*. ICCDE 2020. New York, NY, USA: Association for Computing Machinery, 2020, pp. 247–251. ISBN: 9781450376730. DOI: 10.1145/3379247.3379278. URL: https://doi.org/10.1145/3379247.3379278.

[10] E F E Ahmet, Ula Aslan, and Aytekin Mutlu Kara. “Securing Vulnerabilities in Docker Images”. In: *International Journal of Innovative Engineering Applications* 4.1 (2020), pp. 31–39.

[11] Ulrich Matchi Aïvodji, Sébastien Gambs, and Alexandre Martin. “IOTFLA : A Secured and Privacy-Preserving Smart Home Architecture Implementing

Federated Learning". In: *2019 IEEE Security and Privacy Workshops (SPW)*. 2019, pp. 175–180. DOI: 10.1109/SPW.2019.00041.

[12] Angga Maulana Akbar et al. "Optimizing Software Defect Prediction Models: Integrating Hybrid Grey Wolf and Particle Swarm Optimization for Enhanced Feature Selection with Popular Gradient Boosting Algorithm". In: *Journal of Electronics, Electromedical Engineering, and Medical Informatics* 6.2 (2024), pp. 169–181.

[13] Takuya Akiba et al. "Optuna: A next-generation hyperparameter optimization framework". In: *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery \& data mining*. 2019, pp. 2623–2631.

[14] Yanal Alahmad, Tariq Daradkeh, and Anjali Agarwal. "Availability-Aware Container Scheduler for Application Services in Cloud". In: *2018 IEEE 37th International Performance Computing and Communications Conference (IPCCC)*. 2018, pp. 1–6. DOI: 10.1109/PCCC.2018.8711295.

[15] Yanal Alahmad, Tariq Daradkeh, and Anjali Agarwal. "Optimized Availability-Aware Component Scheduler for Applications in Container-Based Cloud". In: *2019 Sixth International Conference on Software Defined Systems (SDS)*. 2019, pp. 194–199. DOI: 10.1109/SDS.2019.8768654.

[16] Waleed Albattah and Musaad Alzahrani. "Software Defect Prediction Based on Machine Learning and Deep Learning Techniques: An Empirical Approach". In: *AI* 5.4 (Sept. 2024), pp. 1743–1758. ISSN: 2673-2688. DOI: 10.3390/ai5040086.

[17] Anes Aliyu Aihong, Usman ali Ali, and Abuzairu Ahmad. *An Optimized Deep Learning Method for Software Defect Prediction Using Whale optimization Algorithm (WOA): A Review*. Tech. rep. 2023. URL: https://www.researchgate.net/publication/374618693.

[18] Tahir Alyas et al. “Container Performance and Vulnerability Management for Container Security Using Docker Engine”. In: *Security and Communication Networks* 2022 (2022).

[19] Mohammad Mohammadi Amiri and Deniz Gündüz. “Federated learning over wireless fading channels”. In: *IEEE Transactions on Wireless Communications* 19.5 (2020), pp. 3546–3557.

[20] Fan Ang et al. “Robust federated learning with noisy communication”. In: *IEEE Transactions on Communications* 68.6 (2020), pp. 3452–3464.

[21] Davide Anguita et al. “A public domain dataset for human activity recognition using smartphones.” In: *Esann*. Vol. 3. 2013, p. 3.

[22] A Asensio et al. “On the optimality of Concurrent Container Clusters Scheduling over heterogeneous smart environments”. In: *Future Generation Computer Systems* 118 (2021), pp. 157–169. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2021.01.003. URL: https://www.sciencedirect.com/science/article/pii/S0167739X21000054.

[23] Wissal Attaoui et al. “VNF and Container Placement: Recent Advances and Future Trends”. In: *arXiv preprint arXiv:2204.00178* (2022).

[24] Naylor G Bachiega et al. “Container-Based Performance Evaluation: A Survey and Challenges”. In: *2018 IEEE International Conference on Cloud Engineering (IC2E)*. 2018, pp. 398–403. DOI: 10.1109/IC2E.2018.00075.

[25] Rabindra K Barik et al. “Performance analysis of virtual machines and containers in cloud computing”. In: *2016 International Conference on Computing, Communication and Automation (ICCCA)*. 2016, pp. 1204–1210. DOI: 10.1109/CCAA.2016.7813925.

[26] V.R. Basili, L.C. Briand, and W.L. Melo. “A validation of object-oriented design metrics as quality indicators”. In: *IEEE Transactions on Software Engineering* 22.10 (Oct. 1996), pp. 751–761. ISSN: 0098-5589. DOI: 10.1109/32.544352.

[27] Ouafa Bentaleb et al. “Containerization technologies: taxonomies, applications and challenges”. In: *The Journal of Supercomputing* 78.1 (2022), pp. 1144–1181. ISSN: 1573-0484. DOI: 10.1007/s11227-021-03914-1. URL: https://doi.org/10.1007/s11227-021-03914-1.

[28] Aditya Bhardwaj and C Rama Krishna. “A Container-Based Technique to Improve Virtual Machine Migration in Cloud Computing”. In: *IETE Journal of Research* 0.0 (2019), pp. 1–16. DOI: 10.1080/03772063.2019.1605848. URL: https://doi.org/10.1080/03772063.2019.1605848.

[29] Aditya Bhardwaj and C Rama Krishna. “Virtualization in Cloud Computing: Moving from Hypervisor to Containerization—A Survey”. In: *Arabian Journal for Science and Engineering* 46.9 (2021), pp. 8585–8601. ISSN: 2191-4281. DOI: 10.1007/s13369-021-05553-3. URL: https://doi.org/10.1007/s13369-021-05553-3.

[30] Debarshi Bhattacharya et al. “Ensem-HAR: An Ensemble Deep Learning Model for Smartphone Sensor-Based Human Activity Recognition for Measurement of Elderly Health Monitoring”. In: *Biosensors* 12.6 (June 2022). ISSN: 20796374. DOI: 10.3390/bios12060393.

[31] Jing Bi et al. “Integrated deep learning method for workload and resource prediction in cloud systems”. In: *Neurocomputing* 424 (2021), pp. 35–48.

[32] Jing Bi et al. “Integrated deep learning method for workload and resource prediction in cloud systems”. In: *Neurocomputing* 424 (Feb. 2021), pp. 35–48. ISSN: 09252312. DOI: 10.1016/j.neucom.2020.11.011.

[33] Rok Blagus and Lara Lusa. “SMOTE for high-dimensional class-imbalanced data”. In: *BMC Bioinformatics* 14.1 (Dec. 2013), p. 106. ISSN: 1471-2105. DOI: 10.1186/1471-2105-14-106.

[34] Harikrishna Bommala, Rajanikanth Aluvalu, Swapna Mudrakola, et al. “Machine learning job failure analysis and prediction model for the cloud environment”. In: *High-Confidence Computing* 3.4 (2023), p. 100165.

[35] *Bonnie++ - Wikipedia*. URL: https://en.wikipedia.org/wiki/Bonnie%2B%2B.

[36] Rolando Brondolin, Tommaso Sardelli, and Marco D Santambrogio. “DEEPMon: Dynamic and Energy Efficient Power Monitoring for Container-Based Infrastructures”. In: *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. 2018, pp. 676–684. DOI: 10.1109/IPDPSW.2018.00110.

[37] Emiliano Casalicchio and Vanessa Perciballi. “Measuring Docker Performance: What a Mess!!!” In: *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion*. ICPE ’17 Companion. New York, NY, USA: Association for Computing Machinery, 2017, pp. 11–16. ISBN: 9781450348997. DOI: 10.1145/3053600.3053605. URL: https://doi.org/10.1145/3053600.3053605.

[38] MinSu Chae, HwaMin Lee, and Kiyeol Lee. “A performance comparison of linux containers and virtual machines using Docker and KVM”. In: *Cluster Computing* 22.1 (2019), pp. 1765–1775. ISSN: 1573-7543. DOI: 10.1007/s10586-017-1511-2. URL: https://doi.org/10.1007/s10586-017-1511-2.

[39] Sravan Kumar Challa, Akhilesh Kumar, and Vijay Bhaskar Semwal. “A multibranch CNN-BiLSTM model for human activity recognition using wearable sensor data”. In: *The Visual Computer* 38.12 (2022), pp. 4095–4109.

[40] Ngoc-Tu Chau et al. “AppPACK: A Packaging Model for Single-Purpose Lightweight Virtualization Environment”. In: *IEEE Access* 9 (2021), pp. 30071–30079.

[41] Feifei Chen, Xiaofeng Zhou, and Chao Shi. “The Container Scheduling Method Based on the Min-Min in Edge Computing”. In: *Proceedings of the 2019 4th International Conference on Big Data and Computing*. ICBDC 2019. New York, NY, USA: Association for Computing Machinery, 2019, pp. 83–90. ISBN: 9781450362788. DOI: 10.1145/3335484.3335506. URL: https://doi.org/10.1145/3335484.3335506.

[42] Haisong Chen, Linlin Yang, and Aili Wang. “Efficient Cross-Project Software Defect Prediction Based on Federated Meta-Learning”. In: *Electronics* 13.6 (2024), p. 1105.

[43] Jinfu Chen et al. “Software defect prediction approach based on a diversity ensemble combined with neural network”. In: *IEEE Transactions on Reliability* (2024).

[44] Wei Chen et al. “Semi-supervised learning based tag recommendation for docker repositories”. In: *Journal of Computer Science and Technology* 34.5 (2019), pp. 957–971.

[45] Zhijia Chen et al. “Self-Adaptive Prediction of Cloud Resource Demands Using Ensemble Model and Subtractive-Fuzzy Clustering Based Fuzzy Neural Network”. In: *Computational Intelligence and Neuroscience* 2015 (2015), pp. 1–14. ISSN: 1687-5265. DOI: 10.1155/2015/919805.

[46] Kennedy Chengeta. “Comparing the Performance between Virtual Machines and Containers Using Deep Learning Credit Models”. In: *Proceedings of the International Conference on Artificial Intelligence and Its Applications*. icARTi ’21. New York, NY, USA: Association for Computing Machinery, 2021. ISBN: 9781450385756. DOI: 10.1145/3487923.3487934. URL: https://doi.org/10.1145/3487923.3487934.

[47] Mehdi Nazari Cheraghlou, Ahmad Khadem-Zadeh, and Majid Haghparast. “A survey of fault tolerance architecture in cloud computing”. In: *Journal of Network and Computer Applications* 61 (2016), pp. 81–92.

[48] Ron C Chiang. “Contention-aware container placement strategy for docker swarm with machine learning based clustering algorithms”. In: *Cluster Computing* (2020), pp. 1–11.

[49] Garvit Rajesh Choudhary et al. “Empirical analysis of change metrics for software fault prediction”. In: *Computers \& Electrical Engineering* 67 (2018), pp. 15–24.

[50] P Bharath Kumar Chowdary et al. “Deep Learning-Based Forecasting of Task Failures in Cloud Data Centers”. In: *Proceedings of the Fourth International Conference on Advances in Computer Engineering and Communication Systems (ICACECS 2023)*. Vol. 18. Springer Nature. 2023, p. 170.

[51] *Cilium - Linux Native, API-Aware Networking and Security for Containers*. URL: https://cilium.io/.

[52] Jürgen Cito et al. “An empirical analysis of the docker container ecosystem on github”. In: *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE. 2017, pp. 323–333.

[53] Lorenzo Civolani, Guillaume Pierre, and Paolo Bellavista. “FogDocker: Start container now, fetch image later”. In: *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing*. 2019, pp. 51–59.

[54] *Cloud Native & Secure DevOps Platform — Sysdig*. URL: https://sysdig.com/platform-architecture/.

[55] *Container Monitoring — Datadog*. URL: https://www.datadoghq.com/product/container-monitoring/.

[56] *Container monitoring — Dynatrace*. URL: https://www.dynatrace.com/solutions/container-monitoring/.

[57] *Container Monitoring Tool – Monitor Docker Containers — AppOptics*. URL: https://www.appoptics.com/use-cases/container-monitoring.

[58] *Container Monitoring Tools — Sematext*. URL: https://sematext.com/container-monitoring/.

[59] *Container Monitoring: Essential Tools + Best Practices — Scout APM Blog*. URL: https://scoutapm.com/blog/container-monitoring.

[60] *Container Security — Qualys*. URL: https://www.qualys.com/apps/container-security/.

[61] *Container Vulnerability Scanning • Anchore*. URL: https://anchore.com/container-vulnerability-scanning.

[62] Thomas Cover and Peter Hart. "Nearest neighbor pattern classification". In: *IEEE transactions on information theory* 13.1 (1967), pp. 21–27.

[63] Federico Cruciani et al. "Feature learning for human activity recognition using convolutional neural networks: A case study for inertial measurement unit and audio data". In: *CCF Transactions on Pervasive Computing and Interaction* 2.1 (2020), pp. 18–32.

[64] Ali Dabba, Abdelkamel Tari, and Samy Meftali. "A novel grey wolf optimization algorithm based on geometric transformations for gene selection and cancer classification". In: *The Journal of Supercomputing* 80.4 (2024), pp. 4808–4840.

[65] *dbench*. URL: https://dbench.samba.org/doc/dbench.1.html.

[66] Karel Dejaeger, Thomas Verbraken, and Bart Baesens. "Toward comprehensible software fault prediction models using bayesian network classifiers". In: *IEEE Transactions on Software Engineering* 39.2 (2012), pp. 237–257.

[67] Janez Demšar. “Statistical comparisons of classifiers over multiple data sets”. In: *The Journal of Machine learning research* 7 (2006), pp. 1–30.

[68] Qiqing Deng et al. “A Secure Container Placement Strategy Using Deep Reinforcement Learning in Cloud”. In: *2022 IEEE 25th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*. IEEE. 2022, pp. 1299–1304.

[69] Thomas G. Dietterich. “Ensemble Methods in Machine Learning”. In: 2000, pp. 1–15. DOI: 10.1007/3-540-45014-9{_}1.

[70] *Docker Image/Container Security Scan with Clair — Installation — by Kinjal Rathod — System Weakness*. URL: https://systemweakness.com/docker-image-container-security-scan-with-clair-installation-355f80201ef5.

[71] *docker stats — Docker Documentation*. URL: https://docs.docker.com/engine/reference/commandline/stats/.

[72] *Documentation · Wiki · AppArmor / apparmor · GitLab*. URL: https://gitlab.com/apparmor/apparmor/-/wikis/Documentation.

[73] Piotr Dziurzanski and Leandro Soares Indrusiak. “Value-Based Allocation of Docker Containers”. In: *2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*. 2018, pp. 358–362. DOI: 10.1109/PDP2018.2018.00064.

[74] Guisheng Fan et al. *Multi-objective optimization of container-based microservice scheduling in edge computing*. 2020. DOI: 10.2298/CSIS200229041F.

[75] Ebubeogu Amarachukwu Felix and Sai Peck Lee. “Predicting the number of defects in a new software version”. In: *PLOS ONE* 15.3 (Mar. 2020), e0229131. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0229131.

[76] Jie Feng et al. “PMF: A Privacy-Preserving Human Mobility Prediction Framework via Federated Learning”. In: *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 4.1 (Mar. 2020). DOI: 10.1145/3381006. URL: https://doi.org/10.1145/3381006.

[77] Jiechao Gao, Haoyu Wang, and Haiying Shen. “Task failure prediction in cloud data centers using deep learning”. In: *IEEE transactions on services computing* 15.3 (2020), pp. 1411–1422.

[78] Xing Gao et al. “ContainerLeaks: Emerging Security Threats of Information Leakages in Container Clouds”. In: *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 2017, pp. 237–248. DOI: 10.1109/DSN.2017.49.

[79] *Geekbench 5 - Cross-Platform Benchmark*. URL: https://www.geekbench.com/.

[80] Vasundhara Ghate. “Hybrid deep learning approaches for smartphone sensor-based human activity recognition”. In: *Multimedia Tools and Applications* (2021), pp. 1–20.

[81] Sara Gholami, Hamzeh Khazaei, and Cor-Paul Bezemer. “Should you upgrade official docker hub images in production environments?” In: *2021 IEEE/ACM 43rd International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*. IEEE. 2021, pp. 101–105.

[82] Sukhpal Singh Gill and Rajkumar Buyya. “A Taxonomy and Future Directions for Sustainable Cloud Computing”. In: *ACM Computing Surveys* 51.5 (Sept. 2019), pp. 1–33. ISSN: 0360-0300. DOI: 10.1145/3241038.

[83] Sukhpal Singh Gill and Rajkumar Buyya. “Failure management for reliable cloud computing: a taxonomy, model, and future directions”. In: *Computing in Science \& Engineering* 22.3 (2018), pp. 52–63.

[84] *GitHub - cruvolo/ramspeed-smp: RAMspeed/SMP, a cache and memory benchmarking tool*. URL: https://github.com/cruvolo/ramspeed-smp.

[85] *GitHub - eliasgranderubio/dagda: a tool to perform static analysis of known vulnerabilities, trojans, viruses, malware & other malicious threats in docker images/containers and to monitor the docker daemon and running docker containers for detecting anomalous activities*. URL: https://github.com/eliasgranderubio/dagda/.

[86] Xinjie Guan et al. "Application Oriented Dynamic Resource Allocation for Data Centers Using Docker Containers". In: *IEEE Communications Letters* 21.3 (2017), pp. 504–507. DOI: 10.1109/LCOMM.2016.2644658.

[87] Carlos Guerrero, Isaac Lera, and Carlos Juiz. "Genetic algorithm for multi-objective optimization of container allocation in cloud architecture". In: *Journal of Grid Computing* 16.1 (2018), pp. 113–135.

[88] Jayanth Gummaraju, Tarun Desikan, and Yoshio Turner. "Over 30% of official images in docker hub contain high priority security vulnerabilities". In: *Technical Report* (2015).

[89] Yanghu Guo and Wenbin Yao. "A container scheduling strategy based on neighborhood division in micro service". In: *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*. 2018, pp. 1–6. DOI: 10.1109/NOMS.2018.8406285.

[90] Mansi Gupta, Kumar Rajnish, and Vandana Bhattacharjee. "Impact of Parameter Tuning for Optimizing Deep Neural Network Models for Predicting Software Faults". In: *Scientific Programming* 2021 (2021). ISSN: 10589244. DOI: 10.1155/2021/6662932.

[91] Chongomweru Halimu, Asem Kasem, and S H Shah Newaz. “Empirical comparison of area under ROC curve (AUC) and Mathew correlation coefficient (MCC) for evaluating machine learning algorithms on imbalanced datasets for binary classification”. In: *Proceedings of the 3rd international conference on machine learning and soft computing*. 2019, pp. 1–6.

[92] Nils Y Hammerla, Shane Halloran, and Thomas Plötz. “Deep, convolutional, and recurrent models for human activity recognition using wearables”. In: *arXiv preprint arXiv:1604.08880* (2016).

[93] Mubin Ul Haque and M Ali Babar. “Well begun is half done: An empirical study of exploitability \& impact of base-image vulnerabilities”. In: *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE. 2022, pp. 1066–1077.

[94] Sanjay Hardikar, Pradeep Ahirwar, and Sameer Rajan. “Containerization: Cloud Computing based Inspiration Technology for Adoption through Docker and Kubernetes”. In: *2021 Second International Conference on Electronics and Sustainable Communication Systems (ICESC)*. 2021, pp. 1996–2003. DOI: 10.1109/ICESC51422.2021.9532917.

[95] Nima Shiri Harzevili et al. “Characterizing and understanding software security vulnerabilities in machine learning libraries”. In: *2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)*. IEEE. 2023, pp. 27–38.

[96] Foyzul Hassan, Rodney Rodriguez, and Xiaoyin Wang. “Rudsea: recommending updates of dockerfiles via software environment analysis”. In: *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. 2018, pp. 796–801.

[97] Aurelien Havet et al. “Genpack: A generational scheduler for cloud data centers”. In: *2017 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE. 2017, pp. 95–104.

[98] Yingzhe He et al. “Towards security threats of deep learning systems: A survey”. In: *IEEE Transactions on Software Engineering* 48.5 (2020), pp. 1743–1770.

[99] Nikolas Roman Herbst et al. “Self-adaptive workload classification and forecasting for proactive resource provisioning”. In: *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*. New York, NY, USA: ACM, Apr. 2013, pp. 187–198. ISBN: 9781450316361. DOI: 10.1145/2479871.2479899.

[100] Fabio Hernández et al. “Human activity recognition on smartphones using a bidirectional LSTM network”. In: *2019 XXII symposium on image, signal processing and artificial vision (STSIVA)*. IEEE. 2019, pp. 1–5.

[101] Matti Holopainen. “Monitoring Container Environment with Prometheus and Grafana”. In: (2021).

[102] David W. Hosmer, Stanley Lemeshow, and Rodney X. Sturdivant. *Applied Logistic Regression*. Wiley, Mar. 2013. ISBN: 9780470582473. DOI: 10.1002/9781118548387.

[103] Yang Hu, Cees De Laat, and Zhiming Zhao. “Multi-objective container deployment on heterogeneous clusters”. In: *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE. 2019, pp. 592–599.

[104] Yang Hu et al. “Concurrent container scheduling on heterogeneous clusters with multi-resource constraints”. In: *Future Generation Computer Systems* 102 (2020), pp. 562–573. ISSN: 0167-739X. DOI: https://doi.org/10.

1016/j.future.2019.08.025. URL: https://www.sciencedirect.com/science/article/pii/S0167739X19309446.

[105] Delu Huang et al. “Security Analysis and Threats Detection Techniques on Docker Container”. In: *2019 IEEE 5th International Conference on Computer and Communications (ICCC)*. 2019, pp. 1214–1220. DOI: 10.1109/ICCC47050.2019.9064441.

[106] Md Hasan Ibrahim, Mohammed Sayagh, and Ahmed E Hassan. “Too many images on DockerHub! How different are images for the same system?” In: *Empirical Software Engineering* 25.5 (2020), pp. 4250–4281.

[107] Md Hasan Ibrahim, Mohammed Sayagh, and Ahmed E Hassan. “Too many images on DockerHub! How different are images for the same system?” In: *Empirical Software Engineering* 25.5 (Sept. 2020), pp. 4250–4281. ISSN: 1382-3256. DOI: 10.1007/s10664-020-09873-0.

[108] Mahmoud Imdoukh, Imtiaz Ahmad, and Mohammad Alfailakawi. “Optimizing scheduling decisions of container management tool using many-objective genetic algorithm”. In: *Concurrency and Computation: Practice and Experience* 32.5 (2020), e5536. DOI: https://doi.org/10.1002/cpe.5536. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.5536.

[109] *Iozone Filesystem Benchmark*. URL: https://www.iozone.org/.

[110] *iPerf - The TCP, UDP and SCTP network bandwidth measurement tool*. URL: https://iperf.fr/.

[111] Tariqul Islam and Dakshnamoorthy Manivannan. “Predicting application failure in cloud: A machine learning approach”. In: *2017 IEEE International Conference on Cognitive Computing (ICCC)*. IEEE. 2017, pp. 24–31.

[112] Manar Jammal et al. “Availability analysis of cloud deployed applications”. In: *2016 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE. 2016, pp. 234–235.

[113] Mohammad S Jassas and Qusay H Mahmoud. “A failure prediction model for large scale cloud applications using deep learning”. In: *2021 IEEE International Systems Conference (SysCon)*. IEEE. 2021, pp. 1–8.

[114] Mohammad S Jassas and Qusay H Mahmoud. “Failure characterization and prediction of scheduling jobs in google cluster traces”. In: *2019 IEEE 10th GCC Conference \& Exhibition (GCC)*. IEEE. 2019, pp. 1–7.

[115] Omar Javed and Salman Toor. *Understanding the Quality of Container Security Vulnerability Detection Tools*. 2021. DOI: 10.48550/ARXIV.2101.03844. URL: https://arxiv.org/abs/2101.03844.

[116] Devki Nandan Jha et al. “A Holistic Evaluation of Docker Containers for Interfering Microservices”. In: *2018 IEEE International Conference on Services Computing (SCC)*. 2018, pp. 33–40. DOI: 10.1109/SCC.2018.00012.

[117] Shujian Ji, Kejiang Ye, and Cheng-Zhong Xu. “CMonitor: A Monitoring and Alarming Platform for Container-Based Clouds”. In: *Cloud Computing – CLOUD 2019*. Ed. by Dilma Da Silva, Qingyang Wang, and Liang-Jie Zhang. Cham: Springer International Publishing, 2019, pp. 324–339. ISBN: 978-3-030-23502-4.

[118] Congfeng Jiang et al. “Characterizing Co-Located Workloads in Alibaba Cloud Datacenters”. In: *IEEE Transactions on Cloud Computing* 10.4 (Oct. 2022), pp. 2381–2397. ISSN: 2168-7161. DOI: 10.1109/TCC.2020.3034500.

[119] Rodi Jolak et al. “CONSERVE: A framework for the selection of techniques for monitoring containers security”. In: *Journal of Systems and Software* 186 (2022), p. 111158. ISSN: 0164-1212. DOI: https://doi.org/10.1016/

j.jss.2021.111158. URL: https://www.sciencedirect.com/science/article/pii/S0164121221002478.

[120] Ann Mary Joy. "Performance comparison between Linux containers and virtual machines". In: *2015 International Conference on Advances in Computer Engineering and Applications*. 2015, pp. 342–346. DOI: 10.1109/ICACEA.2015.7164727.

[121] Johnsymol Joy and Mercy Paul Selvan. "A comprehensive study on the performance of different Multi-class Classification Algorithms and Hyperparameter Tuning Techniques using Optuna". In: *2022 International Conference on Computing, Communication, Security and Intelligent Systems (IC3SIS)*. IEEE. 2022, pp. 1–5.

[122] Marian Jureczko and Lech Madeyski. "Software product metrics used to build defect prediction models". In: *Report SPR* 2 (2011), p. 2014.

[123] Chanwit Kaewkasi and Kornrathak Chuenmuneewong. "Improvement of container scheduling for Docker using Ant Colony Optimization". In: *2017 9th International Conference on Knowledge and Smart Technology (KST)*. 2017, pp. 254–259. DOI: 10.1109/KST.2017.7886112.

[124] Vikram Kumar Kamboj. "A novel hybrid PSO–GWO approach for unit commitment problem". In: *Neural Computing and Applications* 27.6 (Aug. 2016), pp. 1643–1655. ISSN: 0941-0643. DOI: 10.1007/s00521-015-1962-4.

[125] Supriya Kamthania. "A Novel Deep Learning RBM Based Algorithm for Securing Containers". In: *2019 IEEE International WIE Conference on Electrical and Computer Engineering (WIECON-ECE)*. 2019, pp. 1–7. DOI: 10.1109/WIECON-ECE48653.2019.9019985.

[126] Md Ebtidaul Karim et al. “BHyPreC: a novel Bi-LSTM based hybrid recurrent neural network model to predict the CPU workload of cloud virtual machine”. In: *IEEE Access* 9 (2021), pp. 131476–131495.

[127] Oren Katz, Dror Rawitz, and Danny Raz. “Containers Resource Allocation in Dynamic Cloud Environments”. In: *2021 IFIP Networking Conference (IFIP Networking)*. 2021, pp. 1–9. DOI: 10.23919/IFIPNetworking52078.2021.9472812.

[128] Kuljeet Kaur et al. “EnLoB: Energy and Load Balancing-Driven Container Placement Strategy for Data Centers”. In: *2019 IEEE Globecom Workshops (GC Wkshps)*. 2019, pp. 1–6. DOI: 10.1109/GCWkshps45667.2019.9024592.

[129] Kuljeet Kaur et al. “KEIDS: Kubernetes-Based Energy and Interference Driven Scheduler for Industrial IoT in Edge-Cloud Ecosystem”. In: *IEEE Internet of Things Journal* 7.5 (2020), pp. 4228–4237. DOI: 10.1109/JIOT.2019.2939534.

[130] Faiza Khan et al. “Hyper-parameter optimization of classifiers, using an artificial immune network and its application to software bug prediction”. In: *IEEE Access* 8 (2020), pp. 20954–20964. ISSN: 21693536. DOI: 10.1109/ACCESS.2020.2968362.

[131] Saad Ahmad Khan et al. “Automatic Migration-Enabled Dynamic Resource Management for Containerized Workload”. In: *IEEE Systems Journal* (2022).

[132] Mst Alema Khatun, Mohammad Abu Yousuf, and Mohammad Ali Moni. “Deep CNN-GRU Based Human Activity Recognition with Automatic Feature Extraction Using Smartphone and Wearable Sensors”. In: *2023 International Conference on Electrical, Computer and Communication Engineering (ECCE)*. IEEE. 2023, pp. 1–6.

[133] Nasraldeen Alnor Adam Khleel and Károly Nehéz. "A novel approach for software defect prediction using CNN and GRU based on SMOTE Tomek method". In: *Journal of Intelligent Information Systems* 60.3 (June 2023), pp. 673–707. ISSN: 0925-9902. DOI: 10.1007/s10844-023-00793-1.

[134] Nasraldeen Alnor Adam Khleel and Károly Nehéz. "Software defect prediction using a bidirectional LSTM network combined with oversampling techniques". In: *Cluster Computing* 27.3 (2024), pp. 3615–3638.

[135] Serhat Kiliçarslan. "PSO+ GWO: a hybrid particle swarm optimization and Grey Wolf optimization based Algorithm for fine-tuning hyper-parameters of convolutional neural networks for Cardiovascular Disease Detection". In: *Journal of Ambient Intelligence and Humanized Computing* 14.1 (2023), pp. 87–97.

[136] Youngwook Kim and Brian Toomajian. "Hand gesture recognition using micro-Doppler signatures with convolutional neural network". In: *IEEE Access* 4 (2016), pp. 7125–7130.

[137] Medha Kirti, Ashish Kumar Maurya, and Rama Shankar Yadav. "Fault-tolerance approaches for distributed and cloud computing environments: A systematic review, taxonomy and future directions". In: *Concurrency and Computation: Practice and Experience* 36.13 (2024), e8081.

[138] Barbara Ann Kitchenham and Stuart Charters. *Guidelines for performing Systematic Literature Reviews in Software Engineering*. Tech. rep. EBSE 2007-001. Keele University and Durham University Joint Report, 2007. URL: https://www.elsevier.com/__data/promis_misc/525444systematicreviewsg pdf.

[139] Satoshi Kobayashi et al. "MarNASNets: Towards CNN model architectures specific to sensor-based human activity recognition". In: *IEEE Sensors Journal* (2023).

[140] Ayu Komalasari and Muhammad Zuhri Catur Candra. “Improving Defect Prediction Using Combination of Software Metrics”. In: *2022 International Conference on Data and Software Engineering (ICoDSE)*. IEEE. 2022, pp. 89–94.

[141] Zhanibek Kozhirbayev and Richard O Sinnott. “A performance comparison of container-based technologies for the Cloud”. In: *Future Generation Computer Systems* 68 (2017), pp. 175–182. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2016.08.025. URL: https://www.sciencedirect.com/science/article/pii/S0167739X16303041.

[142] Jitendra Kumar, Rimsha Goomer, and Ashutosh Kumar Singh. “Long Short Term Memory Recurrent Neural Network (LSTM-RNN) Based Workload Forecasting Model For Cloud Datacenters”. In: *Procedia Computer Science* 125 (2018), pp. 676–682. ISSN: 18770509. DOI: 10.1016/j.procs.2017.12.087.

[143] Jitendra Kumar, Rimsha Goomer, and Ashutosh Kumar Singh. “Long Short Term Memory Recurrent Neural Network (LSTM-RNN) Based Workload Forecasting Model For Cloud Datacenters”. In: *Procedia Computer Science* 125 (2018), pp. 676–682. ISSN: 18770509. DOI: 10.1016/j.procs.2017.12.087.

[144] Jitendra Kumar, Rimsha Goomer, and Ashutosh Kumar Singh. “Long short term memory recurrent neural network (LSTM-RNN) based workload forecasting model for cloud datacenters”. In: *Procedia Computer Science* 125 (2018), pp. 676–682.

[145] Jitendra Kumar and Ashutosh Kumar Singh. “An Efficient Machine Learning Approach for Virtual Machine Resource Demand Prediction”. In: *International Journal of Advanced Science and Technology* 123 (Feb. 2019), pp. 21–30. ISSN: 20054238. DOI: 10.33832/ijast.2019.123.03.

[146] Jitendra Kumar and Ashutosh Kumar Singh. “Performance evaluation of metaheuristics algorithms for workload prediction in cloud environment”. In: *Applied Soft Computing* 113 (Dec. 2021), p. 107895. ISSN: 15684946. DOI: 10.1016/j.asoc.2021.107895.

[147] Jitendra Kumar and Ashutosh Kumar Singh. “Workload prediction in cloud using artificial neural network and adaptive differential evolution”. In: *Future Generation Computer Systems* 81 (Apr. 2018), pp. 41–52. ISSN: 0167739X. DOI: 10.1016/j.future.2017.10.047.

[148] Jitendra Kumar, Ashutosh Kumar Singh, and Rajkumar Buyya. “Ensemble learning based predictive framework for virtual machine resource request prediction”. In: *Neurocomputing* 397 (July 2020), pp. 20–30. ISSN: 09252312. DOI: 10.1016/j.neucom.2020.02.014.

[149] Jitendra Kumar et al. “BiPhase adaptive learning-based neural network model for cloud datacenter workload forecasting”. In: *Soft Computing* 24.19 (Oct. 2020), pp. 14593–14610. ISSN: 1432-7643. DOI: 10.1007/s00500-020-04808-9.

[150] Prabhat Kumar and S Suresh. “FLAAP: An Open Human Activity Recognition (HAR) Dataset for Learning and Finding the Associated Activity Patterns”. In: *Procedia Computer Science* 212 (2022), pp. 64–73.

[151] Jennifer R Kwapisz, Gary M Weiss, and Samuel A Moore. “Activity recognition using cell phone accelerometers”. In: *ACM SigKDD Explorations Newsletter* 12.2 (2011), pp. 74–82.

[152] Soonhong Kwon and Jong-Hyouk Lee. “DIVDS: Docker Image Vulnerability Diagnostic System”. In: *IEEE Access* 8 (2020), pp. 42666–42673. DOI: 10.1109/ACCESS.2020.2976874.

[153] Junghye Lee et al. “Privacy-preserving patient similarity learning in a federated environment: development and analysis”. In: *JMIR medical informatics* 6.2 (2018), e7744.

[154] Kyungwoon Lee et al. “Analysis on network performance of container virtualization on IoT devices”. In: *2017 International Conference on Information and Communication Technology Convergence (ICTC)*. 2017, pp. 35–37. DOI: 10.1109/ICTC.2017.8190937.

[155] Lingguang Lei et al. “SPEAKER: Split-phase execution of application containers”. In: *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer. 2017, pp. 230–251.

[156] Tiia Leppänen. “Data visualization and monitoring with Grafana and Prometheus”. In: (2021).

[157] Feng Li, Wen Jun Tan, and Wentong Cai. “A wholistic optimization of containerized workflow scheduling and deployment in the cloud–edge environment”. In: *Simulation Modelling Practice and Theory* 118 (2022), p. 102521.

[158] Jiawei Li et al. “An Improved Container Scheduling Algorithm Based on PSO for Big Data Applications”. In: *Cyberspace Safety and Security*. Ed. by Jaideep Vaidya, Xiao Zhang, and Jin Li. Cham: Springer International Publishing, 2019, pp. 516–530. ISBN: 978-3-030-37337-5.

[159] Juan Li, Hongji Xu, and Yuhao Wang. “Multi-resolution Fusion Convolutional Network for Open Set Human Activity Recognition”. In: *IEEE Internet of Things Journal* (2023).

[160] Li Li et al. “A review of applications in federated learning”. In: *Computers \& Industrial Engineering* 149 (2020), p. 106854.

[161] Lianwan Li, Jianxin Chen, and Wuyang Yan. “A Particle Swarm Optimization-Based Container Scheduling Algorithm of Docker Platform”. In: *Proceedings of the 4th International Conference on Communication and Information Processing*. ICCIP ’18. New York, NY, USA: Association for Computing Machinery, 2018, pp. 12–17. ISBN: 9781450365345. DOI: 10.1145/3290420.3290432. URL: https://doi.org/10.1145/3290420.3290432.

[162] Qilong Li and Yu Fang. “Multi-Algorithm Collaboration Scheduling Strategy for Docker Container”. In: *2017 International Conference on Computer Systems, Electronics and Control (ICCSEC)*. IEEE. 2017, pp. 1367–1371.

[163] Suyi Li et al. “Abnormal client behavior detection in federated learning”. In: *arXiv preprint arXiv:1910.09933* (2019).

[164] Zheng Li et al. “Performance Overhead Comparison between Hypervisor and Container Based Virtualization”. In: *2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA)*. 2017, pp. 955–962. DOI: 10.1109/AINA.2017.79.

[165] Changyuan Lin, Sarah Nadi, and Hamzeh Khazaei. “A large-scale data set and an empirical study of docker images hosted on docker hub”. In: *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE. 2020, pp. 371–381.

[166] Miao Lin et al. “Ant Colony Algorithm for Multi-Objective Optimization of Container-Based Microservice Scheduling in Cloud”. In: *IEEE Access* 7 (2019), pp. 83088–83100. DOI: 10.1109/ACCESS.2019.2924414.

[167] Xin Lin et al. “A Measurement Study on Linux Container Security: Attacks and Countermeasures”. In: *Proceedings of the 34th Annual Computer Security Applications Conference*. ACSAC ’18. New York, NY, USA: Association for Computing Machinery, 2018, pp. 418–429. ISBN: 9781450365697. DOI: 10.

1145/3274694.3274720. URL: https://doi.org/10.1145/3274694.3274720.

[168] Yuhang Lin, Olufogorehan Tunde-Onadele, and Xiaohui Gu. "CDL: Classified Distributed Learning for Detecting Security Attacks in Containerized Applications". In: *Annual Computer Security Applications Conference*. ACSAC '20. New York, NY, USA: Association for Computing Machinery, 2020, pp. 179–188. ISBN: 9781450388580. DOI: 10.1145/3427228.3427236. URL: https://doi.org/10.1145/3427228.3427236.

[169] *LINPACK*. URL: http://www.netlib.org/linpack/.

[170] Bingwei Liu, Yinan Lin, and Yu Chen. "Quantitative workload analysis and prediction using Google cluster traces". In: *2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, Apr. 2016, pp. 935–940. ISBN: 978-1-4673-9955-5. DOI: 10.1109/INFCOMW.2016.7562213.

[171] Bo Liu et al. "A new container scheduling algorithm based on multi-objective optimization". In: *Soft Computing* 22.23 (2018), pp. 7741–7752.

[172] Bo Liu et al. "K-PSO: An improved PSO-based container scheduling algorithm for big data applications". In: *International Journal of Network Management* 31.2 (2021), e2092. DOI: https://doi.org/10.1002/nem.2092. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/nem.2092.

[173] Chunhong Liu et al. "Predicting of job failure in compute cloud based on online extreme learning machine: a comparative study". In: *IEEE Access* 5 (2017), pp. 9359–9368.

[174] Dianbo Liu, Dmitriy Dligach, and Timothy Miller. "Two-stage federated phenotyping and patient representation learning". In: *Proceedings of the confer-*

*ence. Association for Computational Linguistics. Meeting*. Vol. 2019. NIH Public Access. 2019, p. 283.

[175] Peiyu Liu et al. "Understanding the Security Risks of Docker Hub". In: *Computer Security – ESORICS 2020*. Ed. by Liqun Chen et al. Cham: Springer International Publishing, 2020, pp. 257–276. ISBN: 978-3-030-58951-6.

[176] Yu Liu et al. "Performance Evaluation of Containerization in Edge-Cloud Computing Stacks for Industrial Applications: A Client Perspective". In: *IEEE Open Journal of the Industrial Electronics Society* 2 (2021), pp. 153–168. DOI: 10.1109/OJIES.2021.3055901.

[177] Saiqin Long et al. "A Global Cost-Aware Container Scheduling Strategy in Cloud Data Centers". In: *IEEE Transactions on Parallel and Distributed Systems* 33.11 (2022), pp. 2752–2766. ISSN: 15582183. DOI: 10.1109/TPDS.2021.3133868.

[178] Fotis Loukidis-Andreou et al. "Docker-Sec: A Fully Automated Container Security Enhancement Mechanism". In: *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. 2018, pp. 1561–1564. DOI: 10.1109/ICDCS.2018.00169.

[179] Weizhi Lu, Boyu Li, and Bin Wu. "Overhead Aware Task Scheduling for Cloud Container Services". In: *2019 IEEE 23rd International Conference on Computer Supported Cooperative Work in Design (CSCWD)*. 2019, pp. 380–385. DOI: 10.1109/CSCWD.2019.8791871.

[180] Zhigang Lu et al. "An empirical case study on the temporary file smell in dockerfiles". In: *IEEE Access* 7 (2019), pp. 63650–63659.

[181] Yee Jia Luwe, Chin Poo Lee, and Kian Ming Lim. "Wearable sensor-based human activity recognition with hybrid deep learning model". In: *Informatics*. Vol. 9. 3. MDPI. 2022, p. 56.

[182] Jingze Lv, Mingchang Wei, and Yang Yu. “A Container Scheduling Strategy Based on Machine Learning in Microservice Architecture”. In: *2019 IEEE International Conference on Services Computing (SCC)*. 2019, pp. 65–71. DOI: 10.1109/SCC.2019.00023.

[183] Liang Lv et al. “Communication-aware container placement and reassignment in large-scale internet data centers”. In: *IEEE Journal on Selected Areas in Communications* 37.3 (2019), pp. 540–555.

[184] Lech Madeyski and Marian Jureczko. “Which process metrics can significantly improve defect prediction models? An empirical study”. In: *Software Quality Journal* 23.3 (Sept. 2015), pp. 393–422. ISSN: 15731367. DOI: 10.1007/S11219-014-9241-7.

[185] R Madhumathi. “The Relevance of Container Monitoring Towards Container Intelligence”. In: *2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*. 2018, pp. 1–5. DOI: 10.1109/ICCCNT.2018.8493766.

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

[187] Ruchika Malhotra and Anjali Bansal. “Use of Support Vector Machine to Check Whether Process Metrics are as Good as Static Code Metrics”. In: *Topical Drifts in Intelligent Computing: Proceedings of International Conference on Computational Techniques and Applications (ICCTA 2021)*. Vol. 426. Springer Nature. 2022, p. 35.

[188] Ruchika Malhotra, Anjali Bansal, and Marouane Kessentini. “Deployment and performance monitoring of docker based federated learning framework for software defect prediction”. In: *Cluster Computing* (2024), pp. 1–19.

[189] Ruchika Malhotra, Anjali Bansal, and Marouane Kessentini. “Deployment and performance monitoring of docker based federated learning framework for software defect prediction”. In: *Cluster Computing* 27.5 (Aug. 2024), pp. 6039–6057. ISSN: 15737543. DOI: 10.1007/s10586-024-04266-0.

[190] Ruchika Malhotra, Anjali Bansal, and Marouane Kessentini. “Vulnerability Analysis of Docker Hub Official Images and Verified Images”. In: *2023 IEEE International Conference on Service-Oriented System Engineering (SOSE)*. IEEE. 2023, pp. 150–155.

[191] Ruchika Malhotra and Kishwar Khan. “A novel software defect prediction model using two-phase grey wolf optimisation for feature selection”. In: *Cluster Computing* (Dec. 2024). ISSN: 15737543. DOI: 10.1007/s10586-024-04599-w.

[192] Hassaan Malik et al. “DMFL_Net: A Federated Learning-Based Framework for the Classification of COVID-19 from Multiple Chest Diseases Using X-rays”. In: *Sensors* 23.2 (2023), p. 743.

[193] Harri Manninen, Vesa Jääskeläinen, and Jan Olaf Blech. “Performance Evaluation of Containerization Platforms for Control and Monitoring Devices”. In: *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. Vol. 1. 2020, pp. 1061–1064. DOI: 10.1109/ETFA46521.2020.9211901.

[194] Ying Mao et al. “Draps: Dynamic and resource-aware placement scheme for docker containers in a heterogeneous cluster”. In: *2017 IEEE 36th International Performance Computing and Communications Conference (IPCCC)*. IEEE. 2017, pp. 1–8.

[195] A Martin et al. “Docker ecosystem – Vulnerability Analysis”. In: *Computer Communications* 122 (2018), pp. 30–43. ISSN: 0140-3664. DOI: https://

doi.org/10.1016/j.comcom.2018.03.011. URL: https://www.sciencedirect.com/science/article/pii/S0140366417300956.

[196] Ilias Mavridis and Helen Karatza. "Combining containers and virtual machines to enhance isolation and extend functionality on cloud computing". In: *Future Generation Computer Systems* 94 (2019), pp. 674–696. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2018.12.035. URL: https://www.sciencedirect.com/science/article/pii/S0167739X18305764.

[197] Ilias Mavridis and Helen Karatza. "Performance and Overhead Study of Containers Running on Top of Virtual Machines". In: *2017 IEEE 19th Conference on Business Informatics (CBI)*. Vol. 02. 2017, pp. 32–38. DOI: 10.1109/CBI.2017.69.

[198] Hemant Kumar Mehta et al. "WattsApp: Power-Aware Container Scheduling". In: *2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC)*. 2020, pp. 79–90. DOI: 10.1109/UCC48980.2020.00027.

[199] João Mendes-Moreira et al. "Ensemble approaches for regression". In: *ACM Computing Surveys* 45.1 (Nov. 2012), pp. 1–40. ISSN: 0360-0300. DOI: 10.1145/2379776.2379786.

[200] Tianhui Meng, Zhihao Shang, and Katinka Wolter. "An empirical performance and security evaluation of android container solutions". In: *2017 International Conference on Cyber Security And Protection Of Digital Services (Cyber Security)*. 2017, pp. 1–8. DOI: 10.1109/CyberSecPODS.2017.8074850.

[201] Tarek Menouer. "KCSS: Kubernetes container scheduling strategy". In: *The Journal of Supercomputing* 77.5 (2021), pp. 4267–4293.

[202] Tarek Menouer, Christophe Cérin, and Étienne Leclercq. "New multi-objectives scheduling strategies in docker swarmkit". In: *International Conference on Al-*

*gorithms and Architectures for Parallel Processing*. Springer. 2018, pp. 103–117.

[203] Tarek Menouer and Patrice Darmon. "Containers scheduling consolidation approach for cloud computing". In: *International Symposium on Pervasive Systems, Algorithms and Networks*. Springer. 2019, pp. 178–192.

[204] Tarek Menouer and Patrice Darmon. "New scheduling strategy based on multi-criteria decision algorithm". In: *2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. IEEE. 2019, pp. 101–107.

[205] Tarek Menouer et al. "Power efficiency containers scheduling approach based on machine learning technique for cloud computing environment". In: *International Symposium on Pervasive Systems, Algorithms and Networks*. Springer. 2019, pp. 193–206.

[206] *MicroScanner: New Free Image Vulnerability Scanner for Developers - Aqua*. URL: https://www.aquasec.com/news/microscanner-new-free-image-vulnerability-scanner-for-developers/.

[207] Alan Mills, Jonathan White, and Phil Legg. "OGMA: Visualisation for Software Container Security Analysis and Automated Remediation". In: *2022 IEEE International Conference on Cyber Security and Resilience (CSR)*. IEEE. 2022, pp. 76–81.

[208] Seyedali Mirjalili and Andrew Lewis. "The Whale Optimization Algorithm". In: *Advances in Engineering Software* 95 (May 2016), pp. 51–67. ISSN: 09659978. DOI: 10.1016/j.advengsoft.2016.01.008.

[209] Seyedali Mirjalili, Seyed Mohammad Mirjalili, and Andrew Lewis. "Grey Wolf Optimizer". In: *Advances in Engineering Software* 69 (Mar. 2014), pp. 46–61. ISSN: 09659978. DOI: 10.1016/j.advengsoft.2013.12.007.

[210] Ammar Al-Moalmi et al. “A whale optimization system for energy-efficient container placement in data centers”. In: *Expert Systems with Applications* 164 (2021), p. 113719. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2020.113719. URL: https://www.sciencedirect.com/science/article/pii/S0957417420305431.

[211] *Monitoring Docker container metrics using cAdvisor — Prometheus*. URL: https://prometheus.io/docs/guides/cadvisor/.

[212] Roberto Morabito. “Virtualization on Internet of Things Edge Devices With Container Technologies: A Performance Evaluation”. In: *IEEE Access* 5 (2017), pp. 8835–8850. DOI: 10.1109/ACCESS.2017.2704444.

[213] Roberto Morabito, Jimmy Kjällman, and Miika Komu. “Hypervisors vs. Lightweight Virtualization: A Performance Comparison”. In: *2015 IEEE International Conference on Cloud Engineering*. 2015, pp. 386–393. DOI: 10.1109/IC2E.2015.74.

[214] Farnaz Moradi et al. “ConMon: An automated container based network performance monitoring system”. In: *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. 2017, pp. 54–62. DOI: 10.23919/INM.2017.7987264.

[215] Mohd Mustaqeem, Suhel Mustajab, and Mahfooz Alam. “A hybrid approach for optimizing software defect prediction using a grey wolf optimization and multilayer perceptron”. In: *International Journal of Intelligent Computing and Cybernetics* 17.2 (2024), pp. 436–464.

[216] Ronald Mutegeki and Dong Seog Han. “A CNN-LSTM approach to human activity recognition”. In: *2020 international conference on artificial intelligence in information and communication (ICAIIC)*. IEEE. 2020, pp. 362–366.

[217] Preeth E N et al. “Evaluation of Docker containers based on hardware utilization”. In: *2015 International Conference on Control Communication Computing India (ICCC)*. 2015, pp. 697–700. DOI: 10.1109/ICCC.2015.7432984.

[218] Narayan Nahak and Ranjan Kumar Mallick. “Damping of power system oscillations by a novel DE-GWO optimized dual UPFC controller”. In: *Engineering science and technology, an international journal* 20.4 (2017), pp. 1275–1284.

[219] Saurav Nanda and Thomas J Hacker. “RACC: Resource-Aware Container Consolidation Using a Deep Learning Approach”. In: *Proceedings of the First Workshop on Machine Learning for Computing Systems*. MLCS’18. New York, NY, USA: Association for Computing Machinery, 2018. ISBN: 9781450358651. DOI: 10.1145/3217871.3217876. URL: https://doi.org/10.1145/3217871.3217876.

[220] Arijit Nandi, Fatos Xhafa, and Rohit Kumar. “A Docker-based federated learning framework design and deployment for multi-modal data stream classification”. In: *Computing* (2023), pp. 1–35.

[221] Matteo Nardelli, Christoph Hochreiner, and Stefan Schulte. “Elastic Provisioning of Virtual Machines for Container Deployment”. In: *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion*. ICPE ’17 Companion. New York, NY, USA: Association for Computing Machinery, 2017, pp. 5–10. ISBN: 9781450348997. DOI: 10.1145/3053600.3053602. URL: https://doi.org/10.1145/3053600.3053602.

[222] Abdullah B. Nasser et al. “A Robust Tuned K-Nearest Neighbours Classifier for Software Defect Prediction”. In: 2023, pp. 181–193. DOI: 10.1007/978-3-031-20429-6{_}18.

[223] Ali Bou Nassif et al. “Software defect prediction using learning to rank approach”. In: *Scientific Reports* 13.1 (Nov. 2023), p. 18885. ISSN: 2045-2322. DOI: 10.1038/s41598-023-45915-5.

[224] Shubha Brata Nath et al. “Green Containerized Service Consolidation in Cloud”. In: *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*. 2020, pp. 1–6. DOI: 10.1109/ICC40277.2020.9149173.

[225] *NBench - Wikipedia*. URL: https://en.wikipedia.org/wiki/NBench.

[226] *netperf - networking benchmark software - LinuxLinks*. URL: https://www.linuxlinks.com/netperf/.

[227] *netpipe_web*. URL: https://netpipe.cs.ksu.edu/.

[228] Solmaz Niknam, Harpreet S Dhillon, and Jeffrey H Reed. “Federated learning for wireless communications: Motivation, opportunities, and challenges”. In: *IEEE Communications Magazine* 58.6 (2020), pp. 46–51.

[229] Vivian Noronha et al. “Performance Evaluation of Container Based Virtualization on Embedded Microprocessors”. In: *2018 30th International Teletraffic Congress (ITC 30)*. Vol. 01. 2018, pp. 79–84. DOI: 10.1109/ITC30.2018.00019.

[230] Omogbai Oleghe. “Container Placement and Migration in Edge Computing: Concept and Scheduling Models”. In: *IEEE Access* 9 (2021), pp. 68028–68043. DOI: 10.1109/ACCESS.2021.3077550.

[231] *Overview — Prometheus*. URL: https://prometheus.io/docs/introduction/overview/.

[232] *Parallel BZIP2 (PBZIP2)*. URL: http://compression.ca/pbzip2/.

[233] Eva Patel and Dharmender Singh Kushwaha. “A hybrid CNN-LSTM model for predicting server load in cloud computing”. In: *The Journal of Supercomputing* 78.8 (2022), pp. 1–30.

[234] Arnaldo Pereira Ferreira and Richard Sinnott. “A Performance Evaluation of Containers Running on Managed Kubernetes Services”. In: *2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. 2019, pp. 199–208. DOI: 10.1109/CloudCom.2019.00038.

[235] Jayama Pinnamaneni, S Nagasundari, and Prasad Honnavalli. “Identifying Vulnerabilities in Docker Image Code using ML Techniques”. In: *2022 2nd Asian Conference on Innovation in Technology (ASIANCON)*. IEEE. 2022, pp. 1–5.

[236] Max Plauth, Lena Feinbube, and Andreas Polze. “A Performance Survey of Lightweight Virtualization Techniques”. In: *Service-Oriented and Cloud Computing*. Ed. by Flavio De Paoli, Stefan Schulte, and Einar Broch Johnsen. Cham: Springer International Publishing, 2017, pp. 34–48. ISBN: 978-3-319-67262-5.

[237] U Pongsakorn et al. “Container rebalancing: Towards proactive linux containers placement optimization in a data center”. In: *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*. Vol. 1. IEEE. 2017, pp. 788–795.

[238] Amit M Potdar et al. “Performance Evaluation of Docker Container and Virtual Machine”. In: *Procedia Computer Science* 171 (2020), pp. 1419–1428. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2020.04.152. URL: https://www.sciencedirect.com/science/article/pii/S1877050920311315.

[239] M. Prashanthi and M. Chandra Mohan. “Hybrid Optimization-Based Neural Network Classifier for Software Defect Prediction”. In: *International Journal*

*of Image and Graphics* 24.04 (July 2024). ISSN: 0219-4678. DOI: 10.1142/S0219467824500451.

[240] Yongfeng Qian et al. "Privacy-aware service placement for mobile edge computing via federated learning". In: *Information Sciences* 505 (2019), pp. 562–570. ISSN: 0020-0255. DOI: https://doi.org/10.1016/j.ins.2019.07.069. URL: https://www.sciencedirect.com/science/article/pii/S0020025519306814.

[241] Vaibhav Rastogi et al. "Cimplifier: automatically debloating containers". In: *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. 2017, pp. 476–486.

[242] Vaibhav Rastogi et al. "New directions for container debloating". In: *Proceedings of the 2017 Workshop on Forming an Ecosystem Around Software Transformation*. 2017, pp. 51–56.

[243] Charles Reiss, John Wilkes, and Joseph L Hellerstein. "Google cluster-usage traces: format+ schema". In: *Google Inc., White Paper* 1.1-14 (2011), p. 83.

[244] Irina Rish et al. "An empirical study of the naive Bayes classifier". In: *IJCAI 2001 workshop on empirical methods in artificial intelligence*. Vol. 3. 22. 2001, pp. 41–46.

[245] Leonardo R Rodrigues et al. "Network-Aware Container Scheduling in Multi-Tenant Data Center". In: *2019 IEEE Global Communications Conference (GLOBECOM)*. 2019, pp. 1–6. DOI: 10.1109/GLOBECOM38437.2019.9013128.

[246] Leonardo Rosa Rodrigues et al. "Time-constrained and network-aware containers scheduling in GPU era". In: *Future Generation Computer Systems* 117 (2021), pp. 72–86. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2020.11.014. URL: https://www.sciencedirect.com/science/article/pii/S0167739X20330387.

[247] Andrea Rossi et al. “Forecasting workload in cloud computing: towards uncertainty-aware predictions and transfer learning”. In: *Cluster Computing* 28.4 (Aug. 2025), p. 258. ISSN: 1386-7857. DOI: 10.1007/s10586-024-04933-2.

[248] Bowen Ruan et al. “A Performance Study of Containers in Cloud Environment”. In: *Advances in Services Computing*. Ed. by Guojun Wang, Yanbo Han, and Gregorio Martínez Pérez. Cham: Springer International Publishing, 2016, pp. 343–356. ISBN: 978-3-319-49178-3.

[249] Mufeed Ahmed Naji Saif et al. “Multi Objective Resource Scheduling for Cloud Environment using Ant Colony Optimization Algorithm”. In: *JOURNAL OF ALGEBRAIC STATISTICS* 13.3 (2022), pp. 2798–2809.

[250] Nosayba El-Sayed, Hongyu Zhu, and Bianca Schroeder. “Learning from failure across multiple clusters: A trace-driven approach to understanding, predicting, and mitigating job terminations”. In: *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE. 2017, pp. 1333–1344.

[251] *Sensu Plugins*. URL: https://sensu-plugins.io/.

[252] A Sergeev, E Rezedinova, and A Khakhina. “Stress testing of Docker containers running on a Windows operating system”. In: *Journal of Physics: Conference Series*. Vol. 2339. 1. IOP Publishing. 2022, p. 12010.

[253] Monaal Sethi et al. “AttnHAR: Human Activity Recognition using Data Collected from Wearable Sensors”. In: *2023 6th International Conference on Information Systems and Computer Networks (ISCON)*. IEEE. 2023, pp. 1–6.

[254] Jyoti Shetty, Rahul Sajjan, and G Shobha. “Task resource usage analysis and failure prediction in cloud”. In: *2019 9th International Conference on Cloud Computing, Data Science \& Engineering (Confluence)*. IEEE. 2019, pp. 342–348.

[255] Zhenwu Shi et al. “HPKS: High Performance Kubernetes Scheduling for Dynamic Blockchain Workloads in Cloud Computing”. In: *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*. IEEE. 2021, pp. 456–466.

[256] Rui Shu, Xiaohui Gu, and William Enck. “A Study of Security Vulnerabilities on Docker Hub”. In: *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*. CODASPY ’17. New York, NY, USA: Association for Computing Machinery, 2017, pp. 269–280. ISBN: 9781450345231. DOI: 10.1145/3029806.3029832. URL: https://doi.org/10.1145/3029806.3029832.

[257] Ashutosh Kumar Singh et al. “A Quantum Approach Towards the Adaptive Prediction of Cloud Workloads”. In: *IEEE Transactions on Parallel and Distributed Systems* 32.12 (Dec. 2021), pp. 2893–2905. ISSN: 1045-9219. DOI: 10.1109/TPDS.2021.3079341.

[258] Narinder Singh and Hanaa Hachimi. “A New Hybrid Whale Optimizer Algorithm with Mean Strategy of Grey Wolf Optimizer for Global Optimization”. In: *Mathematical and Computational Applications* 23.1 (Mar. 2018), p. 14. DOI: 10.3390/mca23010014.

[259] Nidhi Singh and Shrisha Rao. “Ensemble Learning for Large-Scale Workload Prediction”. In: *IEEE Transactions on Emerging Topics in Computing* 2.2 (June 2014), pp. 149–165. ISSN: 2168-6750. DOI: 10.1109/TETC.2014.2310455.

[260] *Software Composition Analysis Tool - JFrog Xray*. URL: https://jfrog.com/xray/.

[261] *Software Defect Prediction*. URL: https://madeyski.e-informatyka.pl/tools/software-defect-prediction/.

[262] Shengbo Song et al. “Gaia scheduler: A kubernetes-based scheduler framework”. In: *2018 IEEE Intl Conf on Parallel \& Distributed Processing with Applications, Ubiquitous Computing \& Communications, Big Data \& Cloud Computing, Social Computing \& Networking, Sustainable Computing \& Communications (ISPA/IUCC/BDCloud/SocialCom/SustainC*. IEEE. 2018, pp. 252–259.

[263] *SPEC CPU® 2006*. URL: https://www.spec.org/cpu2006/.

[264] *STREAM Benchmark - AMD*. URL: https://developer.amd.com/spack/stream-benchmark/.

[265] Sari Sultan, Imtiaz Ahmad, and Tassos Dimitriou. “Container Security: Issues, Challenges, and the Road Ahead”. In: *IEEE Access* 7 (2019), pp. 52976–52996. DOI: 10.1109/ACCESS.2019.2911732.

[266] *Sysbench - Gentoo Wiki*. URL: https://wiki.gentoo.org/wiki/Sysbench#Using_the_CPU_workload.

[267] Byungchul Tak et al. “Security Analysis of Container Images Using Cloud Analytics Framework”. In: *Web Services – ICWS 2018*. Ed. by Hai Jin, Qingyang Wang, and Liang-Jie Zhang. Cham: Springer International Publishing, 2018, pp. 116–133. ISBN: 978-3-319-94289-6.

[268] Boxiong Tan, Hui Ma, and Yi Mei. “A Genetic Programming Hyper-heuristic Approach for Online Resource Allocation in Container-Based Clouds”. In: *AI 2018: Advances in Artificial Intelligence*. Ed. by Tanja Mitrovic, Bing Xue, and Xiaodong Li. Cham: Springer International Publishing, 2018, pp. 146–152. ISBN: 978-3-030-03991-2.

[269] Boxiong Tan, Hui Ma, and Yi Mei. “A Hybrid Genetic Programming Hyper-Heuristic Approach for Online Two-level Resource Allocation in Container-

based Clouds". In: *2019 IEEE Congress on Evolutionary Computation (CEC)*. 2019, pp. 2681–2688. DOI: 10.1109/CEC.2019.8790220.

[270] Boxiong Tan, Hui Ma, and Yi Mei. "A NSGA-II-based Approach for Multi-objective Micro-service Allocation in Container-based Clouds". In: *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*. 2020, pp. 282–289. DOI: 10.1109/CCGrid49817.2020.00-65.

[271] Boxiong Tan, Hui Ma, and Yi Mei. "Novel genetic algorithm with dual chromosome representation for resource allocation in container-based clouds". In: *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*. IEEE. 2019, pp. 452–456.

[272] Boxiong Tan et al. "A Cooperative Coevolution Genetic Programming Hyper-Heuristic Approach for On-line Resource Allocation in Container-based Clouds". In: *IEEE Transactions on Cloud Computing* (2020), p. 1. DOI: 10.1109/TCC.2020.3026338.

[273] Chakkrit Tantithamthavorn. "Towards a better understanding of the impact of experimental components on defect prediction modelling". In: *Proceedings of the 38th International Conference on Software Engineering Companion*. New York, NY, USA: ACM, May 2016, pp. 867–870. ISBN: 9781450342056. DOI: 10.1145/2889160.2889256.

[274] Ye Tao et al. "Dynamic Resource Allocation Algorithm for Container-Based Service Computing". In: *2017 IEEE 13th International Symposium on Autonomous Decentralized System (ISADS)*. 2017, pp. 61–67. DOI: 10.1109/ISADS.2017.20.

[275] Tengku Nazmi Tengku Asmawi, Azlan Ismail, and Jun Shen. "Cloud failure prediction based on traditional machine learning and deep learning". In: *Journal of Cloud Computing* 11.1 (2022), p. 47.

[276] Jörg Thalheim et al. “Cntr: Lightweight Containers”. In: *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. 2018, pp. 199–212.

[277] Suroj Thapa et al. “Software Defect Prediction Using Atomic Rule Mining and Random Forest”. In: *2020 5th International Conference on Innovative Technologies in Intelligent Systems and Industrial Applications (CITISIA)*. IEEE, Nov. 2020, pp. 1–8. ISBN: 978-1-7281-9437-0. DOI: 10.1109/CITISIA50690.2020.9371797.

[278] *The noploop CPU Benchmark*. URL: https://www.brendangregg.com/blog/2014-04-26/the-noploop-cpu-benchmark.html.

[279] Nikhil Saji Thomas and S. Kaliraj. “An Improved and Optimized Random Forest Based Approach to Predict the Software Faults”. In: *SN Computer Science* 5.5 (May 2024), p. 530. ISSN: 2661-8907. DOI: 10.1007/s42979-024-02764-x.

[280] Aparna Tomar et al. “Docker Security: A Threat Model, Attack Taxonomy and Real-Time Attack Scenario of DoS”. In: *2020 10th International Conference on Cloud Computing, Data Science Engineering (Confluence)*. 2020, pp. 150–155. DOI: 10.1109/Confluence47617.2020.9058115.

[281] “Too many images on DockerHub! How different are images for the same system”. In: ().

[282] Ha-Vu Tran et al. “Lightwave power transfer for federated learning-based wireless networks”. In: *IEEE Communications Letters* 24.7 (2020), pp. 1472–1476.

[283] *Trivy Open Source Vulnerability Scanner — Aqua*. URL: https://www.aquasec.com/products/trivy/.

[284] Olufogorehan Tunde-Onadele et al. “A Study on Container Vulnerability Exploit Detection”. In: *2019 IEEE International Conference on Cloud Engineering (IC2E)*. 2019, pp. 121–127. DOI: 10.1109/IC2E.2019.00026.

[285] Adrian-Ioan Tuns and Adrian Sp\uataru. “Cloud Service Failure Prediction on Google’s Borg Cluster Traces Using Traditional Machine Learning”. In: *2023 25th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*. IEEE. 2023, pp. 162–169.

[286] Jan N Van Rijn and Frank Hutter. “Hyperparameter importance across datasets”. In: *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery \& data mining*. 2018, pp. 2367–2376.

[287] K Vani and S Sujatha. “A machine learning framework for job failure prediction in cloud using hyper parameter tuned MLP”. In: *2022 Second International Conference on Advanced Technologies in Intelligent Control, Environment, Computing \& Communication Engineering (ICATIECE)*. IEEE. 2022, pp. 1–6.

[288] Praneeth Vepakomma et al. “A-Wristocracy: Deep learning on wrist-worn sensing for recognition of user complex activities”. In: *2015 IEEE 12th International conference on wearable and implantable body sensor networks (BSN)*. IEEE. 2015, pp. 1–6.

[289] Xili Wan et al. “Application deployment using Microservice and Docker containers: Framework and optimization”. In: *Journal of Network and Computer Applications* 119 (2018), pp. 97–109. ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2018.07.003. URL: https://www.sciencedirect.com/science/article/pii/S1084804518302273.

[290] Aili Wang et al. “Heterogeneous Defect Prediction Algorithm Combined With Federated Sparse Compression”. In: *IEEE Access* 11 (2023), pp. 23739–23753.

[291] Aili Wang et al. “Heterogeneous Defect Prediction Based on Federated Reinforcement Learning via Gradient Clustering”. In: *IEEE Access* 10 (2022), pp. 87832–87843.

[292] Aili Wang et al. “Heterogeneous Defect Prediction Based on Federated Transfer Learning via Knowledge Distillation”. In: *IEEE Access* 9 (2022), pp. 29530–29540.

[293] Bo Wang et al. “Performance comparison between hypervisor- and container-based virtualizations for cloud users”. In: *2017 4th International Conference on Systems and Informatics (ICSAI)*. 2017, pp. 684–689. DOI: 10.1109/ICSAI.2017.8248375.

[294] Hao Wang et al. “A software defect prediction method using binary gray wolf optimizer and machine learning algorithms”. In: *Computers and Electrical Engineering* 118 (Aug. 2024). ISSN: 00457906. DOI: 10.1016/j.compeleceng.2024.109336.

[295] Lu Wang et al. “An Efficient Load Prediction-Driven Scheduling Strategy Model in Container Cloud”. In: *International Journal of Intelligent Systems* 2023 (2023).

[296] LuKun Wang and RuYue Liu. “Human activity recognition based on wearable sensor using hierarchical deep LSTM networks”. In: *Circuits, Systems, and Signal Processing* 39 (2020), pp. 837–856.

[297] Xiaofei Wang et al. “In-edge ai: Intelligentizing mobile edge computing, caching and communication by federated learning”. In: *IEEE Network* 33.5 (2019), pp. 156–165.

[298] Yang Wang et al. “A Novel Deep Multifeature Extraction Framework Based on Attention Mechanism Using Wearable Sensor Data for Human Activity Recognition”. In: *IEEE Sensors Journal* 23.7 (2023), pp. 7188–7198.

[299] Yulong Wang et al. “ContainerGuard: A Real-Time Attack Detection System in Container-Based Big Data Platform”. In: *IEEE Transactions on Industrial*

*Informatics* 18.5 (2022), pp. 3327–3336. DOI: 10.1109/TII.2020.3047416.

[300] Yulong Wang et al. "DockerWatch: a two-phase hybrid detection of malware using various static features in container cloud". In: *Soft Computing* (2022), pp. 1–17.

[301] *Welcome Page*. URL: https://www.nuttcp.net/Welcome%20Page.html.

[302] Qizhen Weng et al. "MLaaS in the Wild: Workload Analysis and Scheduling in Large-Scale Heterogeneous GPU Clusters". In: *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. Renton, WA: USENIX Association, Apr. 2022, pp. 945–960. ISBN: 978-1-939133-27-4. URL: https://www.usenix.org/conference/nsdi22/presentation/weng.

[303] Jiang Wenhao and Li Zheng. "Vulnerability analysis and security research of docker container". In: *2020 IEEE 3rd International Conference on Information Systems and Computer Aided Education (ICISCAE)*. IEEE. 2020, pp. 354–357.

[304] *What is container security? — Container Image Security — Snyk*. URL: https://snyk.io/learn/container-security/.

[305] John Wilkes et al. "Google cluster-usage traces v3". In: *Google Inc., Mountain View, CA, USA, Technical Report* (2020).

[306] Ye Wu and Haopeng Chen. "Abp scheduler: Speeding up service spread in docker swarm". In: *2017 IEEE International Symposium on Parallel and Distributed Processing with Applications and 2017 IEEE International Conference on Ubiquitous Computing and Communications (ISPA/IUCC)*. IEEE. 2017, pp. 691–698.

[307] Yiwen Wu. “Exploring the relationship between dockerfile quality and project characteristics”. In: *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Companion Proceedings*. 2020, pp. 128–130.

[308] Yiwen Wu et al. “Characterizing the Occurrence of Dockerfile Smells in Open-Source Software: An Empirical Study”. In: *IEEE Access* 8 (2020), pp. 34127–34139. ISSN: 21693536. DOI: 10.1109/ACCESS.2020.2973750.

[309] Kun Xia, Jianguang Huang, and Hanyu Wang. “LSTM-CNN architecture for human activity recognition”. In: *IEEE Access* 8 (2020), pp. 56855–56866.

[310] Qixue Xiao et al. “Security risks in deep learning implementations”. In: *2018 IEEE Security and privacy workshops (SPW)*. IEEE. 2018, pp. 123–128.

[311] Xiaojing XIE and Shyam S Govardhan. “A Service Mesh-Based Load Balancing and Task Scheduling System for Deep Learning Applications”. In: *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*. 2020, pp. 843–849. DOI: 10.1109/CCGrid49817.2020.00009.

[312] Jiwei Xu et al. “Dockerfile tf smell detection based on dynamic and static analysis methods”. In: *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*. Vol. 1. IEEE. 2019, pp. 185–190.

[313] Minxian Xu et al. “esDNN: Deep Neural Network Based Multivariate Workload Prediction in Cloud Computing Environments”. In: *ACM Transactions on Internet Technology* 22.3 (Aug. 2022), pp. 1–24. ISSN: 1533-5399. DOI: 10.1145/3524114.

[314] Minxian Xu et al. “esDNN: deep neural network based multivariate workload prediction in cloud computing environments”. In: *ACM Transactions on Internet Technology (TOIT)* 22.3 (2022), pp. 1–24.

[315] *y-cruncher - A Multi-Threaded Pi Program*. URL: http://www.numberworld.org/y-cruncher/.

[316] Hiroki Yamamoto et al. "Towards privacy preserving cross project defect prediction with federated learning". In: *2023 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE. 2023, pp. 485–496.

[317] Kejiang Ye et al. "Modeling Application Performance in Docker Containers Using Machine Learning Techniques". In: *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*. 2018, pp. 1–6. DOI: 10.1109/PADSW.2018.8644581.

[318] Kang Yin et al. "D-Tagger: A tag recommendation approach for Docker repositories". In: *Proceedings of the Tenth Asia-Pacific Symposium on Internetware*. 2018, pp. 1–10.

[319] Kang Yin et al. "STAR: a specialized tagging approach for Docker repositories". In: *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE. 2018, pp. 426–435.

[320] Takeshi Yoshimura, Rina Nakazawa, and Tatsuhiro Chiba. "ImageJockey: A Framework for Container Performance Engineering". In: *2020 IEEE 13th International Conference on Cloud Computing (CLOUD)*. 2020, pp. 238–247. DOI: 10.1109/CLOUD49709.2020.00043.

[321] Ahmed Zerouali, Tom Mens, and Coen De Roover. "On the usage of JavaScript, Python and Ruby packages in Docker Hub images". In: *Science of Computer Programming* 207 (2021), p. 102653. ISSN: 0167-6423. DOI: https://doi.org/10.1016/j.scico.2021.102653. URL: https://www.sciencedirect.com/science/article/pii/S0167642321000460.

[322] Ahmed Zerouali et al. “ConPan: A Tool to Analyze Packages in Software Containers”. In: *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. 2019, pp. 592–596. DOI: 10.1109/MSR.2019.00089.

[323] Ahmed Zerouali et al. “On the Impact of Outdated and Vulnerable Javascript Packages in Docker Images”. In: *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 2019, pp. 619–623. DOI: 10.1109/SANER.2019.8667984.

[324] Ahmed Zerouali et al. “On the relation between outdated docker containers, severity vulnerabilities, and bugs”. In: *2019 ieee 26th international conference on software analysis, evolution and reengineering (saner)*. IEEE. 2019, pp. 491–501.

[325] Chen Zhang et al. “A survey on federated learning”. In: *Knowledge-Based Systems* 216 (2021), p. 106775.

[326] Dong Zhang et al. “Container oriented job scheduling using linear programming model”. In: *2017 3rd International Conference on Information Management (ICIM)*. 2017, pp. 174–180. DOI: 10.1109/INFOMAN.2017.7950370.

[327] Liangkang Zhang et al. “A Novel Hybrid Model for Docker Container Workload Prediction”. In: *IEEE Transactions on Network and Service Management* (2023).

[328] Lu Zhang et al. “A real-time intrusion detection system based on OC-SVM for containerized applications”. In: *2021 IEEE 24th International Conference on Computational Science and Engineering (CSE)*. 2021, pp. 138–145. DOI: 10.1109/CSE53436.2021.00029.

[329] Qi Zhang et al. “A Comparative Study of Containers and Virtual Machines in Big Data Environment”. In: *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*. 2018, pp. 178–185. DOI: 10.1109/CLOUD.2018.00030.

[330] Qingchen Zhang et al. “An Efficient Deep Learning Model to Predict Cloud Workload for Industry Informatics”. In: *IEEE Transactions on Industrial Informatics* 14.7 (July 2018), pp. 3170–3178. ISSN: 1551-3203. DOI: 10.1109/TII.2018.2808910.

[331] Qingchen Zhang et al. “An efficient deep learning model to predict cloud workload for industry informatics”. In: *IEEE transactions on industrial informatics* 14.7 (2018), pp. 3170–3178.

[332] Rong Zhang et al. “A Genetic Algorithm-Based Energy-Efficient Container Placement Strategy in CaaS”. In: *IEEE Access* 7 (2019), pp. 121360–121373. DOI: 10.1109/ACCESS.2019.2937553.

[333] Rong Zhang et al. “Container-VM-PM Architecture: A Novel Architecture for Docker Container Placement”. In: *Cloud Computing – CLOUD 2018*. Ed. by Min Luo and Liang-Jie Zhang. Cham: Springer International Publishing, 2018, pp. 128–140. ISBN: 978-3-319-94295-7.

[334] Yang Zhang et al. “An insight into the impact of dockerfile evolutionary trajectories on quality and latency”. In: *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*. Vol. 1. IEEE. 2018, pp. 138–143.

[335] Yongheng Zhao and Yanxia Zhang. “Comparison of decision tree methods for finding active objects”. In: *Advances in Space Research* 41.12 (2008), pp. 1955–1959.

[336] Yu Zhao et al. “Deep residual bidir-LSTM for human activity recognition using wearable sensors”. In: *Mathematical Problems in Engineering* 2018 (2018), pp. 1–13.

[337] Yan Zheng, Weiyu Dong, and Jiangtao Zhao. “ZeroDVS: Trace-Ability and Security Detection of Container Image Based on Inheritance Graph”. In: *2021 IEEE 5th International Conference on Cryptography, Security and Privacy (CSP)*. 2021, pp. 186–192. DOI: 10.1109/CSP51677.2021.9357496.

[338] Jiahong Zhou et al. “SemiTagRec: a semi-supervised learning based tag recommendation approach for Docker repositories”. In: *International Conference on Software and Systems Reuse*. Springer. 2019, pp. 132–148.

[339] Ruiting Zhou, Zongpeng Li, and Chuan Wu. “Scheduling Frameworks for Cloud Container Services”. In: *IEEE/ACM Transactions on Networking* 26.1 (2018), pp. 436–450. DOI: 10.1109/TNET.2017.2781200.

[340] Yiqing Zhou, Jian Wang, and Zeru Wang. “Bearing Faulty Prediction Method Based on Federated Transfer Learning and Knowledge Distillation”. In: *Machines* 10.5 (2022), p. 376.

[341] Kun Zhu et al. “IMDAC: A robust intelligent software defect prediction model via multi-objective optimization and end-to-end hybrid deep learning networks”. In: *Software: Practice and Experience* 54.2 (2024), pp. 308–333.

# Supervisor’s Biography

# Supervisor's Biography



**Prof. Ruchika Malhotra**

Head of the Department & Professor
Department of Software Engineering
Delhi Technological University
Email: ruchikamalhotra@dtu.ac.in

Dr. Ruchika Malhotra is Head of Department and Professor in the Department of Software Engineering, Delhi Technological University, Delhi, India. She has served as Associate Dean in Industrial Research and Development, Delhi Technological University. She was awarded with prestigious Raman Fellowship for pursuing Post doctoral research in Indiana University Purdue University Indianapolis USA. She received her master's and doctorate degree in software engineering from the University School of Information Technology, Guru Gobind Singh Indraprastha University, Delhi, India. She has received IBM Faculty Award 2013. She has been ranked amongst the World's top 2% scientist by Stanford University report, USA, for her research in the field of "Artificial Intelligence & Image Processing" (Ploss Biology, 2020). She is recipient of Commendable Research Award (in 2018-2024) by Delhi Technological University. Her h-index is 39 as reported by Google Scholar. She is author of book titled "Empirical Research in Software Engineering" published by CRC press and co-author of a book on Object Oriented Software Engineering published by PHI Learning. She has published more than 200 research papers in international journals and conferences. Her research interests are in software testing, improving software quality, statistical and adaptive prediction models, software metrics and the definition and validation of software metrics.

# Supervisor's Biography



**Dr. Marouane Kessentini**

Full Professor & Dean
College of Computing
Grand Valley State University
Email: kessentm@gvsu.edu

Dr. Marouane Kessentini is currently a full professor and Dean of the College of Computing at Grand Valley State University in Michigan and director of the NSF IUCRC center on Pervasive AI; Michigan Site. He received his Ph.D. from the University of Montreal in Canada in 2012. He is a recipient of the prestigious 2018 President of Tunisia distinguished research award, the University distinguished teaching award, the University distinguished digital education award, the College of Engineering and Computer Science distinguished research award, 4 best paper awards including an IEEE 10 Year Most Influential Paper award in Program Comprehension (2011-2021), and his AI-based software refactoring invention, licensed and deployed by Fortune 500 companies, and selected as one of the Top 8 inventions at the University of Michigan for 2018 among over 500 inventions, by the UM Technology Transfer Office. He received various grants from both industry and federal agencies and published over 160 papers in top journals and conferences. He has extensive collaborations with the industry on different areas related to refactoring, software engineering intelligence, search-based software engineering, Edge AI, AI/MLOps, AI and cyber-physical systems, intelligent software bots, etc. He is the co-founder of many workshops, General Chair of SSBSE16 and ASE22, and PC chair of MODELS19, SANER 2021, GECCO, etc. He served as a keynote speaker at various venues including ICSR, SSBSE, GECCO, WCCI, etc. He graduated 15+ Ph.D. students and served as associate editor in 7 journals and PC member of over 150 conferences.

# Author's Biography



**Ms. Anjali Bansal**

Assistant Professor
Department of Computer Science and Engineering
Delhi Technological University
Email: anjalibansal@dtu.ac.in

Anjali Bansal is an assistant professor in the Department of Computer Science and Engineering at Delhi Technological University (DTU) and is pursuing a part-time Ph.D. in the Department of Software Engineering at DTU, where her research focuses on the development and evaluation of methodologies for maintaining software containers. She has completed her M.Tech in Software Engineering from DTU and her B.Tech in Computer Science and Engineering from Noida Institute of Engineering and Technology. She has qualified UGC-NET JRF in 2022 and GATE in 2019. Her academic research interests include Machine Learning, Deep Learning, Federated Learning, Predictive Analytics, Software Engineering, Image Processing, and Cloud Computing. She has published research papers in SCI-indexed journals such as ACM Computing Surveys, Cluster Computing and International Journal of Information Security and has presented her work at various national and international conferences.

SPRINGER NATURE Link

Account

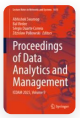
Menu Search Cart

Home > Proceedings of Data Analytics and Management > Conference paper

# A Hybrid Metaheuristic Approach: Whale Optimization and Grey Wolf for Deep Learning in Software Bug Prediction

| Conference paper | First Online: 20 October 2025

| pp 423–435 | Cite this conference paper



Proceedings of Data Analytics and Management

(ICDAM 2025)

Ruchika Malhotra, Anjali Bansal & Marouane Kessentini

Part of the book series: Lecture Notes in Networks and Systems ((LNNS,volume 1615))

Included in the following conference series:
International Conference on Data Analytics & Management

104 Accesses

## Abstract

Predicting bugs in software is necessary for enhancing the quality of software project and reducing maintenance-related risks by identifying problematic software components. It is a methodology for creating a model that can be employed by software professionals and academics throughout the commencement of software development life cycle (SDLC) to detect buggy modules or classes. In this study, we introduced a hybrid metaheuristic swarm intelligence based deep learning model for early-stage prediction of software bugs. Combination of whale optimizer algorithm (WOA) as well as grey wolf optimizer (GWO) is used for optimizing the hyperparameters of Convolutional neural network. The effectiveness of proposed model is esitmated against 6 java-based projects. The proposed model's predictive performance is examined based on several performance measures namely f1 score, MCC, accuracy, precision, AUC, as well as recall. The proposed model's predictive performance is compared with CNN model without HPT, CNN with GWO as HPT approach, and CNN with WOA as HPT approach. The result of this study indicates that the hybrid model of GWO and WOA has significantly improved the predictive performance of CNN model in respect of accuracy, f1 score, AUC, recall, MCC, as well as precision. It is also found that the rankings and superiority of hyperparameter tuning methods is hybrid of GWO and WOA > GWO > WOA.

This is a preview of subscription content, log in via an institution to check access.

Access this chapter

Log in via an institution

Subscribe and save

Springer+ from €37.37 /Month

Starting from 10 chapters or articles per month

Access and download chapters and articles from more than 300k books and 2,500 journals

Cancel anytime

View plans

Buy Now

Chapter EUR 29.95 Price includes VAT (India) eBook EUR 213.99 Softcover Book EUR 249.99

Available as PDF

Read on any device



# A Systematic Literature Review on Maintenance of Software Containers

RUCHIKA MALHOTRA, Delhi Technological University, Delhi, India
ANJALI BANSAL, Delhi Technological University, Delhi, India
MAROUANE KESSENTINI, Oakland University, Rochester, USA

Nowadays, cloud computing is gaining tremendous attention to deliver information via the internet. Virtualization plays a major role in cloud computing as it deploys multiple virtual machines on the same physical machine and thus results in improving resource utilization. Hypervisor-based virtualization and containerization are two commonly used approaches in operating system virtualization. In this article, we provide a systematic literature review on various phases in maintenance of containers including container image detection, container scheduling, container security measures, and performance evaluation of containers. We have selected 145 primary studies out of which 24% of studies are related to container performance evaluation, 42% of studies are related to container scheduling techniques, 22% of studies are related to container security measures, and 12% of studies are related to container image detection process. A few studies are related to container image detection process and evaluation of container security measures. Resource utilization is the most considered performance objective in almost all container scheduling techniques. We conclude that there is a need to introduce new tagging approaches, smell detection approaches, and also new approaches to detect and resolve threat issues in containers so that we can maintain the security of containers.

CCS Concepts: • **General and reference → Surveys and overviews;** • **Software and its engineering → Cloud computing;** • **Theory of computation → Scheduling algorithms;** • **Security and privacy → Virtualization and security;** • **Computing methodologies → Machine learning**;

Additional Key Words and Phrases: Containers, Docker, orchestration, performance evaluation, scheduling techniques, vulnerability, kubernetes

**ACM Reference Format:**
Ruchika Malhotra, Anjali Bansal, and Marouane Kessentini. 2024. A Systematic Literature Review on Maintenance of Software Containers. *ACM Comput. Surv.* 56, 8, Article 193 (April 2024), 38 pages. https://doi.org/10.1145/3645092

## 1 INTRODUCTION

The area of cloud computing is gaining tremendous attention nowadays. It manages resources dynamically and automatically. Without virtualization, there is no existence of cloud computing as it reduces hardware maintenance overhead [36]. We can achieve virtualization with the help of

Authors' addresses: R. Malhotra and A. Bansal, Department of Software Engineering, Delhi Technological University, New Delhi, 110042, India; e-mail: ruchikamalhotra@dtu.ac.in; M. Kessentini, Computer Science Department, Oakland University, Michigan, Rochester, 48309, USA.
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM 0360-0300/2024/04-ART193
https://doi.org/10.1145/3645092

IEEE 2023 @ Greece CISOSE
IEEE INTERNATIONAL CONGRESS ON INTELLIGENT AND SERVICE-ORIENTED SYSTEMS ENGINEERING

IEEE COMPUTER SOCIETY

ΧΑΡΟΚΟΠΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ
HAROKOPIO UNIVERSITY

IEEE FTS
IEEE FUTURE TECHNOLOGY SUMMIT
2023 @ Greece

# 2023 IEEE CISOSE

## *Certificate of Attendance*

**presented to**

*Anjali Bansal*

for attendance and participation of the **2023 IEEE CISOSE International Conference on Inteligent and Service - Oriented Systems Engineering**
IEEE JCC 2023, IEEE SOSE 2023, IEEE BigDataService 2023,
IEEE MobileCloud 2023, IEEE DAPPS 2023, IEEE AI TEST 2023

**July 17-20, 2023**

**Prof. Jerry Gao**
Professor, Research Center Director,
San Jose State University, USA
Executive Chair of Steering Board of
IEEE CISOSE 2023

**Prof. Iraklis Varlamis**
Assoc. Professor
Department of Informatics and Telematics
Harokopio University of Athens
General Chair of IEEE CISOSE 2023

**Prof. Mara Nikolaidou**
Prof. of Servise oriented system engineering
Rector of Harokopio University of Athens
Honorary Summit Organisation
Chair of IEEE CISOSE 2023

# ICDAM-2024

## INTERNATIONAL CONFERENCE ON DATA ANALYTICS & MANAGEMENT (ICDAM-2024)

# Certificate

This is to certify that **Prof. /Dr. / Mr. / Ms. Anjali Bansal** is a presenter/co-author of the paper titled **Towards Federated Learning Approach for Workload Prediction in Cloud Computing** presented at the **5th International Conference on Data Analytics and Management (ICDAM-2024),** organized jointly by London Metropolitan University, London, UK in association with the WSG University, Bydgoszcz, Poland, Europe, Portalegre Polytechnic University, Portugal, Europe, SGGW Management Institute & BPIT, GGSIPU, Delhi on **14th – 15th June 2024.**

**Prof. (Dr.) Bal Virdee**
London Metropolitan University, UK
Convener

**Prof. (Dr.) Abhishek Swaroop**
Bhagwan Parshuram Institute of Technology, India
Publication Chair

# ICDAM-2025

## INTERNATIONAL CONFERENCE ON DATA ANALYTICS & MANAGEMENT (ICDAM - 2025)

## *Certificate*

This is to certify that **Prof. /Dr. / Mr. / Ms. Anjali Bansal** is a presenter/co-author of the paper titled **A Hybrid Metaheuristic Approach: Whale Optimization and Grey Wolf for Deep Learning in Software Bug Prediction** presented at the **6th International Conference on Data Analytics and Management (ICDAM-2025),** organized jointly by London Metropolitan University, London, United Kingdom in association with the WSG University, Bydgoszcz, Poland; Portalegre Polytechnic University, Portugal & SGGW Management Institute, Poland, Europe during **13th – 15th June 2025.**

**Prof. (Dr.) Bal Virdee**
London Metropolitan University, UK
Convener

**Prof. (Dr.) Abhishek Swaroop**
Bhagwan Parshuram Institute of Technology, India
Publication Chair

Cluster Computing (2024) 27:6039–6057
https://doi.org/10.1007/s10586-024-04266-0

# Deployment and performance monitoring of docker based federated learning framework for software defect prediction

Ruchika Malhotra[1] · Anjali Bansal[1] · Marouane Kessentini[2]

Received: 1 September 2023 / Revised: 30 December 2023 / Accepted: 3 January 2024 / Published online: 20 February 2024
© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2024

**Abstract**
There are significant challenges in machine learning models due to information security and data privacy issues. In traditional machine learning approaches, the data used to train the centralized model may be sensitive leading to privacy issues. Federated learning overcomes these issues by following the concept of bringing the code to the data instead of data to the code. In today's era of digital transformation, researchers and industries are focusing on cloud-based technologies. New approaches such as microservice models, docker, kubernetes, containers, and virtual machines are being used by IT industries to develop and deploy applications. We make use of software containers to deploy federated learning models and then we identify security vulnerabilities and resource usage in federated learning model. Flask, docker, cAdvisor, prometheus, and grafana are integrated to monitor the resource usage of the federated learning model in terms of CPU utilization, memory utilization, network traffic usage, and disk usage. The generated federated learning model is deployed on docker, and security vulnerabilities and performance are analyzed using open-source tools. The result shows that the generated model is secure and performs efficiently.

**Keywords** Federated learning · Flask · Docker · Software defect prediction · Monitoring tools · Security vulnerabilities

## 1 Introduction

Software development life cycle consists of software testing which is time and resource-consuming. The testing procedure aims to deliver software that is completely error-free and meets the needs of all stakeholders. Testing adds to the overall project budget because it is an expensive process. The effectiveness and quality of the software improve when faults are correctly predicted at an early stage [1]. Finding software defects is a necessary but expensive process. Many approaches such as machine learning (ML) techniques, ensemble techniques, deep learning (DL) techniques, and data mining techniques have been introduced to predict the defects. Giray et al. [2] performed a literature survey on 102 studies to analyze the use of DL techniques in software defect prediction. They concluded that a large number of studies used supervised DL techniques and CNN is the most used DL technique in software defect prediction.

Although ML and DL approaches have proven their efficiency in all areas but these approaches do not perform well on decentralized data. Federated learning (FL) is gaining significant attention nowadays as it works well on distributed and decentralized data and this approach also maintains privacy of the data. Data security and data privacy is an essential issue that needs to be resolved. In FL, various clients work together to solve ML issues while being coordinated by a central server. The idea of FL has appeared frequently in various forms throughout the history of ML and computer science. For example, collaborative deep learning, privacy-preserving machine learning, federated optimization, distributed deep learning, and privacy-preserving data analytics. FL is a decentralized ML

✉ Anjali Bansal
anjalibansal791@gmail.com

Ruchika Malhotra
ruchikamalhotra@dtu.ac.in

Marouane Kessentini
kessentini@oakland.edu

[1] Department of Software Engineering, Delhi Technological University, New Delhi, India

[2] Computer Science Department, Oakland University, Rochester, MI, USA

International Journal of Information Security (2026) 25:49
https://doi.org/10.1007/s10207-026-01213-5

REGULAR CONTRIBUTION

# Assessing security vulnerabilities in a docker-enabled federated learning framework with hyperparameter tuning for software bug prediction

**Ruchika Malhotra**[1] · **Anjali Bansal**[1] · **Marouane Kessentini**[2]

Received: 31 July 2025 / Accepted: 19 January 2026
© The Author(s) 2026

**Abstract**

Federated Learning (FL) has come up as a promising paradigm for collaborative machine learning over decentralized data by maintaining user privacy at the same time enabling broader data access. Although FL is increasingly being used, its security during deployment, especially in containerized environments, remains under-explored. This study directly addresses the gap by examining security risks within a Docker-based FL deployment designed for Software Bug Prediction (SBP). We developed two federated models—a Convolutional Neural Network (FLCNN) as well as an Artificial Neural Network (FLANN)—and integrated hyperparameter tuning (HPT) using the Grey Wolf Optimizer (GWO). After training the model on multiple open-source SBP datasets, we containerized the best FLCNN model, deploying it as a Flask API within Docker. Vulnerability scans, executed using open-source tools (Anchore, Aqua Trivy, Snyk, JFrog Xray), disclosed several security flaws, including critical and high-severity risks such as denial-of-service, buffer overflow, as well as memory management issues. By subsequently applying updated dependencies and recommended container security protocols, we established a substantially more secure FL deployment profile. This analysis underscores the essential need for rigorous, continuous security assessment when deploying federated learning via containerization.

**Keywords** Hyperparameter tuning · Federated learning · Deep learning · Software bug prediction · Docker

## 1 Introduction

The cost of software development life cycle (SDLC) often rises due to the presence of software bugs, as the expense of fixing errors increases with the delay in their discovery. Since SDLC includes software testing, which demands both significant time and resources, it becomes a major contributor to project costs. The primary aim of testing is to ensure that the software functions correctly and meets stakeholder requirements. This step, though essential, adds to the total development budget. Detecting faults early in the process can substantially improve the effectiveness and quality of the software. Locating software bugs remains a necessary yet costly task. SBP provides a way to identify potentially faulty components—such as modules, classes, files, or methods—before testing begins. By doing so, SBP helps reduce inspection costs and supports quality assurance teams in using their resources more effectively. Integrating SBP within the SDLC improves testing efficiency, enhances software reliability, and minimizes the negative impact of defects on final deliverables. Various computational methods, including machine learning, deep learning, ensemble modeling, and data mining, have been successfully applied to automate the process and improve software quality outcomes. A considerable amount of research has been conducted in the area of SBP, reflecting the field's maturity and continuing relevance. However, there remain unexplored directions for developing privacy-aware and secure approaches to bug prediction. Li et al. [1] reviewed existing SBP studies to outline the key stages of the SBP process and to highlight emerging areas for future investigation. Similarly, Giray et al. [2] examined 102 publications focusing on deep learning methods used in SBP. Their review showed that most studies applied supervised learning techniques, with

Anjali Bansal
anjalibansal791@gmail.com

[1] Department of Software Engineering, Delhi Technological University, New Delhi, India

[2] College of Computing, Grand Valley State University, Allendale, MI, USA

Published online: 09 February 2026

SPRINGER NATURE Link

Log in

Menu Search Cart



# Toward Federated Learning Approach for Workload Prediction in Cloud Computing

| Conference paper | First Online: 13 April 2025

| pp 85–93 | Cite this conference paper



Proceedings of Data Analytics and Management

(ICDAM 2024)

Ruchika Malhotra, Anjali Bansal & Marouane Kessentini

Part of the book series: Lecture Notes in Networks and Systems ((LNNS,volume 1299))

Included in the following conference series:
International Conference on Data Analytics & Management

134 Accesses

## Abstract

Cloud computing (CC) has become a key paradigm in the field of information technology, providing scalable and on demand resource allocation to satisfy the dynamic needs of modern applications. Accurate workload prediction is crucial for addressing dynamic resource allocation and scheduling challenges within cloud computing environment. Although machine learning (ML) and deep learning (DL) approaches are frequently used for workload prediction, their centralized nature increases the risk of data leakage. So, federated learning overcomes the above issues. Federated learning is privacy preserving machine learning paradigm which follows the concept of moving the code to the data rather than moving the data to the code. In this paper, we use Alibaba cluster trace v2018, Alibaba cluster trace 2020, google cloud trace (GCT) 2011, and GCT 2019 for predicting the workload in cloud computing environment by applying federated leaning. The results after applying federated learning to train various deep learning models show high accuracy meanwhile preserving the security and privacy of the data.

This is a preview of subscription content, log in via an institution to check access.

Access this chapter

# Vulnerability Analysis of Docker Hub Official Images and Verified Images

Ruchika Malhotra
Department of Software Engineering
Delhi Technological Univeristy
New Delhi, India
ruchikamalhotra@dtu.ac.in

Anjali Bansal
Department of Software Engineering
*Delhi Technological Univeristy*
New Delhi, India
anjalibansal791@gmail.com

Marouane Kessentini
Department of Computer Science & Engineering
Oakland Univeristy
Rochester, MI
kessentini@oakland.edu

2023 IEEE International Conference on Service-Oriented System Engineering (SOSE) | 979-8-3503-2239-2/23/$31.00 ©2023 IEEE | DOI: 10.1109/SOSE58276.2023.00025

***Abstract*— Container technology is gaining significant attention as compared to virtual machines due to an increase in the use of cloud computing and containers use fewer resources as compared to virtual machines. Docker is the most widely used container technology that helps in managing and running containers. Containers use images for execution that can be created with the help of a docker file or can be downloaded from various open-source repositories. Docker uses a Docker hub repository that consists of official and verified images. As containers share the host operating system, there is a need to monitor the security of the images. In this paper, we are analyzing the vulnerabilities in official and verified docker images with the help of open-source vulnerability detection tools such as anchore, aqua trivy, docker scan and jfrog xray. This paper helps in identifying which types of images are more secure based on the number of vulnerabilities and severity of vulnerabilities and whether the number of pulls and number of stars affects the number of vulnerabilities in images.**

***Keywords*— *Docker, Containers, Images, Security, Vulnerabilities, Tools***

## I. Introduction

The complexity of maintaining and managing application services across large distributed computing environments is minimized by cloud-based infrastructure. As cloud computing is gaining a lot of attention, the use of virtual machine has also increased. Since virtual machines provide a way to run multiple operating system on a single device but virtual machines require a lot of resources from the host. This is where containers come into the picture as containers uses few resources.

Containerization is a technique which is used to bind a software system and its dependencies, binary files, libraries and all the necessary files required to run the software into one package which is called as a container. The main objective of using containers is to deploy a software system on various platform without any platform dependency issues and compatibility issues. Docker is the most popular containerization platform which reduces the deployment time of software system so that developers quickly ship the software into production phase.

Dockerhub is the publicly available repository which consist of docker images (official and verified images). Official images are published by docker team itself and verified images are published by a certified publisher. Dockerhub also provides a service through which developers can share their images in private mode (with selected group of users) or public mode (with whole community of dockerhub users). One can reuse the existing image by pulling it from dockerhub instead of building image from scratch.

As the docker images are maintained and updated by the users, Docker doesn't have any control on how frequently the images (libraries and packages of images) get updated. The images can go months and years without any updates. If any image uses outdated libraries or any outdated packages, this may introduce some vulnerabilities in the images and may harm the security of the containers.

A vulnerability refers to any flaws and weaknesses in the system due to which an attacker can gain control of the system. These vulnerabilities can occur due to two reasons: either there are some errors in code or there is some defect in the design of the system. Vulnerabilities can be of various types such as outdated packages, buffer overflow, SQL injection, security misconfiguration etc. According to OWASP, the most common security vulnerabilities are cryptographic failures, broken access control, insecure design, identification and authentication failures, outdated components, software and data integrity failures, injection etc.

There are two terms docker images and docker containers. Docker images and docker containers are almost same thing. A docker image is a file which is a combination of binary files, library files, and all the necessary dependencies required to run an application. A docker container is a read/write copy of a docker image. When we run a docker image it builds docker container.

The growing use of containers has led to an increased need to examine security related issues in containers. Containers are more tightly integrated with the operating system of the host as compared to virtual machines. Due to this there is a risk that any container can access the sensitive and malicious data from other containers or the host system itself. Therefore, there is a need to examine the security issues in containers

The objective of this study is to analyze the security vulnerabilities in official and verified images of docker hub with the help of various open-source image scanning tools. We have formulated following research questions:

- Is there any clear difference between the total number of vulnerabilities identified by various image scanning tools in official and verified docker images?
- Is there any clear difference between the type of vulnerabilities (severity of vulnerabilities) identified in

2642-6587/23/$31.00 ©2023 IEEE
DOI 10.1109/SOSE58276.2023.00025



Authorized licensed use limited to: University of Michigan-Flint. Downloaded on October 13,2023 at 18:36:07 UTC from IEEE Xplore. Restrictions apply.