IJNRD.ORG ISSN : 2456-4184
INTERNATIONAL JOURNAL OF NOVEL RESEARCH AND DEVELOPMENT (IJNRD) | IJNRD.ORG
An International Open Access, Peer-reviewed, Refereed Journal

"AUTOMATED SOFTWARE BUG DETECTION USING MACHINE LEARNING : A NOVEL APPROACH FOR EFFICIENT SOFTWARE QUALITY ASSURANCE "

NAME : CHINTHA.DURGA RAM CHARAN TEJ.

ADDRESS : GUNADALA,VIJAYAWADA,INDIA

ABSTRACT :

SOFTWARE BUGS CAN LEAD TO SECURITY VULNERABILITES,SYSTEM FAILURES, AND FINANCIAL LOSSES.

TRADITIONAL BUG DETECTION METHODS RELY ON MANUAL CODE REVIEWS AND STATIC ANALYSIS,WHICH ARE TIME -CONSUMING AND ERROR-PRONE.

THIS IS REASERCH PROPOSE AS AUTOMATED BUG DETECTION SYSTEM USING MACHINE LEARNING(ML) TO IMPROVE SOFTWARE QUALITY ASSURANCE.OUR MODEL LEVERAGES

NATURAL LANGUAGE PROCESSING(NLP) AND DEEP LEARNING TO ANALYSE SOURCE CODE AND DETECT POTENTIAL BUGS WITH HIGH ACCURACY.

WE COMPARE DIFFERENT ML ALGORITHMS,SUCH AS RANDOM FOREST,SUPPORT VECTOR MACHINE(SVM),AND DEEP NEURAL NETWORKS,TO DETERMINE THE MOST EFFECTIVE APPROACH.

THE PROPOSED SYSTEM OUT PERFORMS TRADITIONAL METHODS IN TERMS OF PRECISION,RECALL, AND F1-SCORE,MAKING IT A PROMISING SOLUTION FORAUTOMATED SOFTWARE DEBUGGING.

KEYWORDS :

MACHINE LEARNING , BUG DETECTION , SOFTWARE QUALITY , DEEP LEARNING , STATIC CODE ANALYSIS.

1.INTRODUCTION :

- **OVERVIEW OF SOFTWARE BUGS AND THEIR IMPACT ON SOFTWARE RELIABILITY AND SECURITY.**
- **LIMITATIONS OF TRADITIONAL BUG DETECTION METHODS(MANUAL CODE REVIEW , STATIC ANALYSIS TOOLS)**
- **INTRODUCTION TO MACHINE LEARNING FOR AUTOMATED BUG DETECTION.**
- **RESEARCH OBJECTIVES AND CONTRIBUTIONS.**

Reliability and quality of a software has now become a top priority for the maintenance of reputation in the market. The most important part of maintenance is bug detection to maintain the quality of the product.

The later you detect it the more expensive it will become to remove it. Traditional detection methods take a long time in QA testing and usually fail in detecting them from the product due to the complexity.

Then it causes late releases and much more cost. So the advancement is machine learning that helps in the detection and has wonderfully transformed the detection process. So let's learn more about it.

- **TIME CONSUMING :**

In real-world settings, clients do not devote extra time to software development. Both software developers and customers expect that the software development will be completed on time. Then they don't provide extra time for issue handling. It takes a long time to finish the defect Management procedure manually.

Previous research studies have mentioned that when people conduct particular processes, it takes a long time compared to machines.

After evaluating prior studies, we found that the testing stage of the software development

life cycle takes more time to fix or manage defects.

2.LITERATURE REVIEW :

•OVERVIEW OF EXISTING BUG DETECTION TECHNIQUES(e.g,, STATIC/DYNAMIC ANALYSIS,AI-BASED METHODS)

•PREVIOUS ML-BASED BUG DETECTION APPROACHES AND THEIR LIMITATIONS.

•ADVANCES IN NLP FOR SOURCE CODE ANALYSIS.

•GAP IN EXISTING RESEARCH AND JUSTIFICATION FOR PROPOSED METHOD.

•In recent years, numerous Machine Learning (ML) models, including Deep Learning (DL) and classic ML models, have been developed to detect software vulnerabilities. However, there is a notable lack of comprehensive and systematic surveys that summarize, classify, and analyze the applications of these ML models in software vulnerability detection. This absence may lead to critical research areas being overlooked or under-represented, resulting in a skewed understanding of the current state of the art in software vulnerability detection. To close this gap, we propose a comprehensive and systematic literature review that characterizes the different properties of ML-based software vulnerability detection systems using six major Research Questions (RQs).

•Using a custom web scraper, our systematic approach involves extracting a set of studies from four widely used online digital libraries: ACM Digital Library, IEEE Xplore, ScienceDirect, and Google Scholar. We manually analyzed the extracted studies to filter out irrelevant work unrelated to software vulnerability detection, followed by creating taxonomies and addressing RQs. Our analysis indicates a significant upward trend in applying ML techniques for software vulnerability detection over the past few years, with many studies published in recent years. Prominent conference venues include the International Conference on Software Engineering (ICSE), the International Symposium on Software Reliability Engineering (ISSRE), the Mining Software Repositories (MSR) conference, and the ACM International Conference on the Foundations of Software Engineering (FSE), whereas Information and Software Technology (IST), Computers & Security (C&S), and Journal of Systems and Software (JSS) are the leading journal venues.

3. PROPOSED METHODOLOGY :

3.1 DATA COLLECTION & PREPROCESSING :

Code Repositories:

Source Code:

Collect code files (files, classes, methods) along with their characteristics (code metrics, syntax, complexity).

Change History: Track code modifications, including commits, authors, and timestamps, to understand how code evolves and relates to bug occurrences.

Bug Tracking Systems:

Bug Reports: Gather bug descriptions, severity levels, assigned developers, and resolution status.

Bug Context: Include information about the code modules, functions, or files where bugs were found.

Testing Data:

Test Cases: Collect test cases, including inputs, expected outputs, and actual outputs, to identify failures.

Test Results: Store test results, including pass/fail status, error messages, and execution times.

External Data:

Dependencies: Include information about external libraries and dependencies, as bugs can arise from interactions between different components.

Documentation: Analyze documentation, including API documentation, to identify potential ambiguities or inconsistencies.

Public Datasets:

Leverage publicly available datasets like the BugHunter Dataset, PROMISE warehouse, and NASA dataset for benchmarking and validation.

•DATA USED (E.G.BUGZILLA,GITHUB,DEFECTS4J)

•DATA CLEANING AND PREPROCESSING STEPS(TOKENIZATION,FEATURE EXTRACTION)

•HANDLING IMBALANCED DATASETS (OVERSAMPLING,UNDERSAMPLING TECHNIQUES)

3.2 FEATURE EXTRACTION & MODEL SELECTION :

• EXTRACTION MEANINGFUL FEATURES(SYNTAX PATTERNS,FUNCTION CALLS,DEPENDENCE)

•COMPARATIVE ANALYSIS OF ML ALGORITHMS(SVM,DECISION TREES,RANDOM FOREST,DEEP LEARNING)

•SELECTION OF THE BEST-PERFORMING MODEL BASED ON ACCURACY METRICS

•Complexity Metrics: Calculate cyclomatic complexity, method length, and nesting depth.

•Coupling and Cohesion: Analyze the relationships between different code modules.

•Code Size: Measure the size of code elements (files, classes, methods).

3.3 MODEL TRAINING & EVALUATION :

•TRAINING METHODOLOGY (SUPERVISED LEARNING,DEEP LEARNING MODELS)

•PERFORMANCE EVALUATION METRICS : ACCURACY,PRECISION,RECALL,F1-SCORE

•COMPARATIVE RESULTS OF DIFFERENT ML MODELS

•Bug report assignment is an important part of software maintenance. In particular, incorrect assignments of bug reports to development teams can be very expensive in large software development projects. Several studies propose automating bug assignment techniques using machine learning in open source software contexts, but no study exists for large-scale proprietary projects in industry. The goal of this study is to evaluate automated bug assignment techniques that are based on machine learning classification. In particular, we study the state-of-the-art ensemble learner Stacked Generalization (SG) that combines several classifiers. We collect more than 50,000 bug reports from five development projects from two companies in different domains. We implement automated bug assignment and evaluate the performance in a set of controlled experiments. We show that SG scales to large scale industrial application and that it outperforms the use of individual classifiers for bug assignment, reaching prediction accuracies from 50 % to 89 % when large training sets are used. In addition, we show how old training data can decrease the prediction accuracy of bug assignment. We advice industry to use SG for bug assignment in proprietary contexts, using at least 2,000 bug reports for training. Finally, we highlight the importance of not solely relying on results from cross-validation when evaluating automated bug assignment.

3.4 SYSTEM IMPLEMENTATION :

•WORKFLOW OF THE PROPOSED BUG DETECTION SYSTEM.

•INTEGRATION INTO A REAL-WORLD SOFTWARE DEVELOPMENT

ENVIRNOMENT.

•AUTOMATION AND REAL-TIME ANALYSIS CAPABILITIES.

•Abstract—Implementations of stateful security protocols mustcarefully manage the type and order of exchanged messages andcryptographic material, by maintaining a state machine whichkeeps track of protocol progress. Corresponding implementation flaws, called state machine bugs, can constitute serious security vulnerabilities. We present an automated black-box technique for detecting state machine bugs in implementations of stateful network protocols. It takes as input a catalogue of state machine bugs for the protocol, each specified as a finite automaton which accepts sequences of messages that exhibit the bug, and a (possibly inaccurate) model of the implementation under test, typically obtained by model learning. Our technique constructs the set of sequences that (according to the model) can be performed by the implementation and that (according to the automaton) expose the bug. These sequences are then transformed to test cases on the actual implementation to find a witness for the bug or filter out false alarms. We have applied our technique on three widelyused implementations of SSH servers and nine different DTLS server and client implementations, including their most recent versions. Our technique easily reproduced all bugs identified by security researchers before, and produced witnesses for them.More importantly, it revealed several previously unknown bugs in the same implementations, two new vulnerabilities, and a variety of new bugs and non-conformance issues in newer versions of the same SSH and DTLS implementations.

4.EXPERIMENTAL RESULTS & DISCUSSION :

•PERFORMANCE COMPARSION OF DIFFERENT ML MODELS.

•VISUALIZATION OF RESULTS (CONFUSION MATRIX,ROC CURVES)

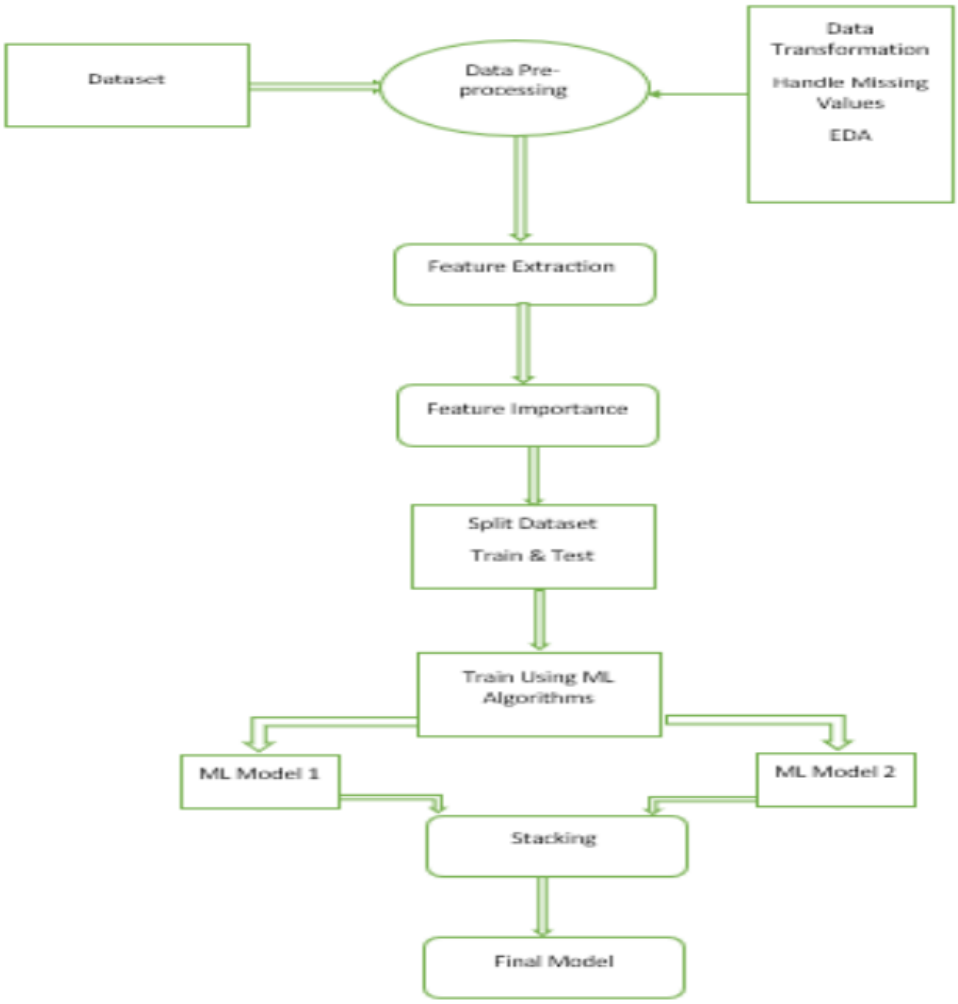
•CASE STUDIES : REAL-WORLD APPLICATION OF THE PROPOSED SYSTEM ON OPEN-SOURCE PROJECTS.

•CHALLENGES FACED AND SOLUTIONS IMPLEMENTED.

•Bug report assignment is an important part of software maintenance. In particular, incorrect assignments of bug reports to development teams can be very expensive in large software development projects. Several studies propose automating bug assignment techniques using machine learning in open source software contexts, but no study exists for large-scale proprietary projects in industry. The goal of this study is to evaluate automated bug assignment techniques that are based on machine learning classification. In particular, we study the state-of-the-art ensemble learner Stacked Generalization (SG) that combines several classifiers. We collect more than 50,000 bug reports from five development projects from two companies in different domains. We implement automated bug assignment and evaluate the

performance in a set of controlled experiments. We show that SG scales to large scale industrial application and that it outperforms the use of individual classifiers for bug assignment, reaching prediction accuracies from 50 % to 89 % when large training sets are used. In addition, we show how old training data can decrease the prediction accuracy of bug assignment. We advice industry to use SG for bug assignment in proprietary contexts, using at least 2,000 bug reports for training. Finally, we highlight the importance of not solely relying on results from cross-validation when evaluating automated bug assignment.

-



-

5. CONCLUSION & FUTURE WORK :

•SUMMARY OF KEYS FINDINGS.

•CONTRINUTIONS TO SOFTWARE QUALITY IMPROVEMENTS

•FUTURE RESEARCH DIRECTIONS (E.G : INTEGRATING REINFORCEMENT LEARNING ,REAL-TIME DEBUGGING IN IDEs)

•By using the sampled data obtained in the exploratory study as part of the dataset for evaluation, we developed an end-to-end tool designed to automatically identify non-functional bug reports in deep learning frameworks. This method capitalizes on semantic knowledge learning, considers hierarchical structures, and employs diverse feature extraction techniques. The primary advantage of this approach is its ability to identify bug reports using advanced neural language model, significantly reducing the need for intensive human analysis. Our approach outperforms nine other leading classifiers, demonstrating substantial improvements with strong statistical significance—specifically, an increase in AUC (Area Under the Curve) by up to 71%. Additionally, it achieved the top Scott-Knott ranking in four frameworks and the second-best in one.

•During our research, we discovered that labeling datasets is both time-consuming and labor-intensive. To streamline this process, we developed a cross-project framework that automates and enhances the identification of bug reports from GitHub repositories through a synergy of human and machine efforts. We assessed MHNurf with a dataset encompassing 1,275,881 reports from over 127,000 software projects, comparing it against contemporary methodologies, basic benchmarks, and various adaptations. Remarkably, MHNurf achieved substantial efficiencies, reducing effort by up to 95.8% for readability and 196.0% for identifiability during human labeling tasks. This approach not only improved performance metrics, such as the F1-score, in bug report identification but also proved to be model-agnostic, enhancing performance across diverse neural language models. Additionally, a qualitative case study involving 10 participants further validated the effectiveness of MHNurf, with users reporting significant savings in time and cost.

•Through the extensive research conducted in this thesis, we have identified and effectively addressed several critical knowledge gaps within the existing literature. Consequently, we have advanced the field of bug report analysis using neural language models by developing more precise, stable, and dependable analysis techniques.