



3(b)Details of Publications in UGC Care-I Journals

Annexure IV-C



Jagan Institute of Management Studies



NAAC A++ Grade & NBA Accredited MCA Program

3, Institutional Area, Sector-5, Rohini, Delhi-110085

Publication in UGC CARE (Category-I) Journal for Academic session 2025-26

1	Dr. KALIKA PATRAI	INDUSTRIAL ENGINEERING JOURNAL	RELIABILITY ANALYSIS OF A THREE UNIT DEGRADABLE COMPUTING SYSTEM WITH COVERAGE FACTOR USING PARTICLE SWARM OPTIMIZATION	UGC CARE GROUP 1	http://www.journal-iiie-india.com/1_jan-june_26.html	ISSN 0970 2555	Jun-26
2	Dr. Deepak Chahal	JOURNAL OF ADVANCE AND FUTURE RESEARCH	Deep Learning-Based Malware Classification Techniques: A Comprehensive Performance Analysis	GOOgle Scholar & UGC CARE	https://rjwave.org/jaafr/papers/JAAFR2606546.pdf	2984-889X	Jun-26
3	Dr. Latika Kharb	JOURNAL OF ADVANCE AND FUTURE RESEARCH	GENERATIVE AI FOR AUTOMATED CODE GENERATION AND TESTING: A COMPREHENSIVE EMPIRICAL STUDY	GOOgle Scholar & UGC CARE	https://rjwave.org/jaafr/papers/JAAFR2606526.pdf	2984-889X	Jun-26



RELIABILITY ANALYSIS OF A THREE-UNIT DEGRADABLE COMPUTING SYSTEM WITH COVERAGE FACTOR USING PARTICLE SWARM OPTIMIZATION

Dr. Kalika Patrai, Associate Professor, Department of Management, Jagan Institute of Management Studies, New Delhi, India

ABSTRACT

This paper provides a reliability analysis for a three-unit degenerative computer system with a coverage factor that works in a non-repairable environment. This system is made up of three processing units that operate in active redundancy. The system degrades gradually through operational states before complete failure. A Continuous-Time Markov Process will be used to formulate the probability model of the system. The Kolmogorov differential equations will then be solved to get the transient state probabilities. The analytical solutions for the reliability of the system and Mean Time to System Failure (MTTF) will be obtained in closed form. Particle Swarm Optimization (PSO) is utilized to optimize system reliability by minimizing failure rate and maximizing the coverage factor.

Keywords: Reliability, Degradable Computing System, Coverage Factor, Markov Process, Particle Swarm Optimization, MTTF.

I. Introduction and Literature Review

Modern computing systems such as distributed computing, cloud computing, industrial automation, communication systems, aerospace systems, and telecommunication networks require high reliability and uninterrupted operation because failures may cause severe operational losses, economic damage, and degradation of system performance. Redundancy techniques are therefore widely employed to improve reliability and fault tolerance in critical computing environments [1]. Reliability engineering has become one of the most significant research areas in computer systems, communication networks, and industrial systems because system failures may lead to catastrophic consequences in real-time applications.

In many real-world systems, failure of an individual component does not immediately lead to total system collapse. Instead, the system continues operating in degraded mode with reduced efficiency and performance. Such systems are known as degradable systems or gracefully degradable systems. Degradable systems are extensively used in cloud computing, multiprocessor systems, distributed architectures, and aviation control systems where uninterrupted service is essential [2]. Shama [3] has study the reliability measures of a degradable system in which switching failures and reboot delay are considered. Failure times and repair times of failed units are assumed to be exponentially distributed. Pandey and Tyagi [4] estimated profust reliability of a powerloom plant, which is modelled as a two unit gracefully degradable system. The Markov process has been widely adopted for modelling degradable systems because it efficiently handles state transitions, degradation behaviour, fault coverage, and stochastic failures. Uprety and Patrai [5] evaluated system characteristics (MTTF, reliability) of a two unit degradable system using Markov modelling approach, in which times to failure and times to repair of the operating units are, assumed to follow fuzzified exponential distribution.

Coverage factor is another important reliability parameter in fault-tolerant computing systems. The coverage factor represents the probability that the system successfully detects, isolates, and recovers from failed components. Higher coverage significantly improves system reliability and survivability. Imperfect coverage may cause catastrophic failure even when redundant units are available. Therefore, incorporation of coverage factor makes the reliability model more practical and realistic [6]. Uprety and Zaheeruddin [7] developed a fuzzy Markov model of gracefully degradable



JAAFR
INTERNATIONAL
RESEARCH JOURNAL

JOURNAL OF ADVANCE AND FUTURE RESEARCH

JAAFR.ORG | ISSN : 2984-889X

An International Open Access, Peer-reviewed, Refereed Journal

DEEP LEARNING-BASED MALWARE CLASSIFICATION TECHNIQUES: A COMPREHENSIVE PERFORMANCE ANALYSIS

Dr. Deepak Chahal

Department of IT, Jagan Institute of Management Studies, Rohini, New Delhi, India.

deepak.chahal@jimsindia.org

Abstract : The rapid proliferation and increasing sophistication of malicious software (malware) pose severe threats to modern computing infrastructure. Traditional signature-based detection mechanisms struggle to keep pace with polymorphic, metamorphic, and zero-day malware variants. Consequently, machine learning—and more recently, deep learning (DL)—has emerged as a paradigm shift in automated malware analysis. This paper provides a comprehensive, rigorous evaluation of deep learning-based malware classification techniques. We investigate three prominent architectural paradigms: Convolutional Neural Networks (CNNs) trained on malware-to-image representations, Recurrent Neural Networks (RNN/LSTM) analyzing sequential API call invocations, and Graph Neural Networks (GNNs) operating on Control Flow Graphs (CFGs). Using the standardized benchmark datasets MalImg and Ember, we conduct a systematic comparative analysis evaluating classification accuracy, computational overhead, and resilience against adversarial evasion [1]. Our empirical results demonstrate that while CNN-based image models offer rapid processing speeds and high accuracy on localized features (98.42% accuracy on MalImg), graph-based architectures exhibit superior generalization capabilities against structural obfuscation at the cost of higher computational complexity.

IndexTerms - Deep Learning, Malware Classification, Feature Extraction, Static Analysis Dynamic Analysis.

I. INTRODUCTION

The modern threat landscape is characterized by an exponential increase in the volume and complexity of cyberattacks. Malware serves as the primary instrument for data breaches, ransomware campaigns, intellectual property theft, and infrastructure disruption. Historically, security analysts relied heavily on static signature matching (e.g., cryptographic hashes like MD5 or SHA-256) and heuristic rules (e.g., YARA rules). While these techniques are computationally efficient and produce low false-positive rates for known threats, they fail completely when encountering zero-day vulnerabilities or binaries obfuscated via packing, encryption, or instruction substitution [2].



JAAFR
INTERNATIONAL
RESEARCH JOURNAL

JOURNAL OF ADVANCE AND FUTURE RESEARCH

JAAFR.ORG | ISSN : 2984-889X

An International Open Access, Peer-reviewed, Refereed Journal

GENERATIVE AI FOR AUTOMATED CODE GENERATION AND TESTING: A COMPREHENSIVE EMPIRICAL STUDY

Dr. Latika Kharb

Department of IT, Jagan Institute of Management Studies, Rohini, New Delhi, India.

latika.kharb@jimsindia.org

Abstract : The integration of Generative Artificial Intelligence (AI) into the Software Development Life Cycle (SDLC) has catalyzed a paradigm shift in how source code is synthesized and validated. Large Language Models (LLMs), particularly those based on Transformer architectures such as GPT-4, StarCoder, and Code Llama, have demonstrated unprecedented capabilities in understanding context, generating functional code, and writing unit tests. This paper presents a comprehensive empirical evaluation of generative AI for automated code generation and testing. We evaluate leading LLMs across standard algorithmic benchmarks (HumanEval, MBPP) and real-world software testing frameworks (Defects4J). Our research investigates two primary dimensions: the functional correctness of generated code and the efficacy of AI-synthesized unit tests in achieving high coverage and revealing synthetic faults (mutation testing). The empirical results demonstrate that while state-of-the-art models achieve up to 82.3% pass@1 on basic algorithmic tasks, their performance degrades on complex, multi-dependency enterprise repositories. Furthermore, while AI-generated tests increase line coverage by an average of 34% compared to human baselines, they frequently struggle with advanced mocking and boundary value edge cases. This paper highlights current capabilities, uncovers critical security and hallucination limitations, and proposes a framework for hybrid human-AI software engineering.

IndexTerms - Generative AI, Large Language Models, Automated Code Generation, Software Testing, Unit Testing, HumanEval, Prompt Engineering.

I. INTRODUCTION

The Software Development Life Cycle (SDLC) is fundamentally constrained by human cognitive limits. Writing code, debugging, and maintaining rigorous test coverage are highly resource-intensive processes. Historically, the software industry has attempted to automate these phases using static analysis tools, Search-Based Software Engineering (SBSE), and template-driven scaffolding. However, these traditional methods are rigidly rule-based, failing to adapt to novel algorithmic challenges or unstructured natural language requirements [1].

The advent of Large Language Models (LLMs) pre-trained on vast corpora of source code (e.g., GitHub repositories, StackOverflow) has introduced a fundamentally new approach: treating code synthesis as a sequence-to-sequence translation problem[2]. By translating natural language specifications (e.g., "Write a Python function to sort an array using quicksort") into executable syntax, models like OpenAI's Codex, Meta's Code Llama, and BigCode's StarCoder are augmenting human developers at an unprecedented scale.

Despite rapid adoption—evidenced by commercial tools like GitHub Copilot—significant gaps remain in the empirical understanding of their reliability, particularly in continuous integration and automated testing. Generating code is only half the software engineering equation; verifying its correctness, security, and robustness is arguably more critical. This paper contributes to the field by:

1. Conducting a systematic performance evaluation of three leading Generative AI models across diverse coding tasks.
2. Assessing the viability of LLMs as automated test-suite generators (Unit and Integration testing).
3. Analyzing the trade-offs between computational overhead, context-window limitations, and functional accuracy.
4. Identifying persistent failure modes, including API hallucinations and subtle security vulnerabilities.