International Journal of Multidisciplinary and Current Research ISSN: 2321-3124
Research Article Available at: http://ijmcr.com

Lightweight Adversarial Software Testing via CNN and Particle Swarm

[1*]Venkata Sivakumar Musam and [2]Karthick.M

[1]Nisum chile, Santiago, Chile
[2]Nandha College of Technology, Erode

Received 01 Jan 2021, Accepted 20 Feb 2021, Available online 22 Feb 2021, **Vol.9 (Jan/Feb 2021 issue)**

Abstract

Convolutional neural networks (CNNs), in particular, are machine-learning models that are increasingly being utilised in modern software development to forecast problems and guide testing procedures. The fundamental issue that this paper addresses is how to conduct effective adversarial software testing in environments where there are hard resource constraints. Current methods whether deep neural networks or exhaustive search—yield high-quality adversarial inputs but require a large amount of computation, memory, and latency. This work presents a lightweight adversarial testing methodology that blends CNNs and PSO to ensure efficient and goal-oriented detection of software vulnerabilities. The innovation here is in its lightweight and efficient nature with a unique combination of Convolutional Neural Networks and Particle Swarm Optimization to conduct adversarial software testing. This integration facilitates efficient defect detection with substantially less computational complexity thus, it is appropriate for real-time and resource-limited scenarios. Experimental outcomes show that the CNN-PSO model is more efficient and effective compared to conventional adversarial testing techniques. On benchmark datasets including PROMISE and NASA MDP, the model recorded an average accuracy of 93.7%, F1 score of 0.91, and adversarial attack success rate of 87.5% while improving computational time reduction to 40% compared to conventional deep learning-based methods. Current adversarial software testing techniques tend to depend on intricate deep learning models or exhaustive search algorithms, which, although efficient, require high computational resources and time. The resultant hybrid solution achieves an interesting balance of precision and performance, demonstrating superiority over existing methods in terms of speed, efficiency, and portability to low-power or embedded platforms at no expense in defect detection efficacy.

Keywords: *Software Testing, Convolutional Neural Network, Particle Swarm Optimization*

1. Introduction

Lightweight Adversarial Software Testing using CNN and Particle Swarm Optimization" deals with improving the robustness of software systems using adversarial testing, with a particular focus on making the testing process lightweight, efficient, and effective [1]. The work proposes a new approach that integrates Convolutional Neural Networks (CNN) with Particle Swarm Optimization (PSO) to produce adversarial test cases for software systems in an efficient and lightweight way [2]. CNN is applied to anticipate possible weak points in the behaviour of the software, and PSO optimally searches through the input space to produce optimized adversarial inputs that can induce faults [3].

The research titled Lightweight Adversarial Software Testing using CNN and Particle Swarm Optimization explores an innovative approach to enhancing the robustness of software systems through adversarial testing [4].

*Corresponding author's ORCID ID : 0000-0003-0816-4774
DOI : https://doi.org/10.14741/ijmcr/v.9.1.2

The primary goal is to make the testing process more efficient, effective, and lightweight, addressing the need for faster and more scalable testing methods in modern software development [5]. The proposed solution combines two powerful techniques—Convolutional Neural Networks (CNN) and Particle Swarm Optimization (PSO)—to generate adversarial test cases that can effectively stress-test software systems [6]. CNN is employed to predict potential vulnerabilities or weak points in the behavior of the software by analyzing its performance under varying inputs [7]. This predictive capability helps identify areas of concern that might otherwise be overlooked during traditional testing processes [8].

High Computational Cost of Traditional Adversarial Testing is Several methods of adversarial testing are dependent on sophisticated optimization algorithms or gradient-based approaches [9] that are expensive in terms of computation. These are not viable in resource-limited systems [10]. These Issues Impacts Efficiency to Lowered Speed the Increased complexity slows down the

generation process, hence not efficient for real-time testing [11]. Resource Drain necessity of heavy computation and memory impacts deployment on light or mobile systems [12]. Inefficiency-led Limited Coverage fewer adversary inputs can be generated and tested within a time frame; thus, vulnerability coverage remains incomplete [13]. These problems focus on the Convolutional Neural Network (CNN) technique and its goal of identifying patterns and learning from the software's execution behaviour [14].

On the other hand, PSO is used to navigate the input space and optimize the generation of adversarial inputs that are specifically designed to uncover faults in the system [15]. By mimicking the way particles in nature explore a space, PSO effectively searches for the most impactful test cases that can induce errors or failures in the software,[16] ensuring that the testing process is not only thorough but also efficient [17]. The integration of CNN with PSO creates a robust framework that significantly reduces the computational overhead associated with traditional adversarial testing, making it more suitable for large-scale or real-time applications [18]. The approach promises to improve the overall reliability of software systems by providing a lightweight yet highly effective method for uncovering hidden defects and vulnerabilities,[19] ultimately leading to more resilient and fault-tolerant software [20].

It assists to CNN is trained to predict regions of the input space where faults or abnormal behaviour are most probable, essentially minimizing the need for testing the whole input space [21]. Limitation of CNN performance greatly relies on the training data quality and quantity and might not generalize for intricate or novel input patterns [22], particularly in highly dynamic software behaviour [23]. Particle Swarm Optimization (PSO) Goal of an optimization algorithm that draws inspiration from bird/fish social behaviour [24]. By searching and iterating over possible test cases from feedback, it helps PSO identify the optimal adversarial test inputs [25]. Its performance is parameter-tuning sensitive (inertia, cognitive and social terms) [26]. Its strength of being fast,adaptive input generator [27]. For increased software testing efficiency and effectiveness, this proposal offers a lightweight automated adversarial software testing technique based on Convolutional Neural Networks (CNN) and Particle Swarm Optimisation (PSO) [28].

The main contributions of the proposed methods using,

- Design a lightweight adversarial testing framework that integrates CNN and PSO to effectively detect software vulnerabilities with minimal computational cost [29].
- Implement an automated mechanism that learns software behaviour patterns using CNN to identify high-risk input regions for targeted testing [30].
- Optimize adversarial test case generation using Particle Swarm Optimization to maximize fault detection efficiency while reducing test generation time [31].

2. Related Works

generative AI in test-case generation and bug detection in automated testing and mention its potential to increase test coverage, effectiveness, and scalability in software testing [32]. Nevertheless, the approach is hampered by data quality, data domain specificity, and continuous human knowledge. system enhancing AI-based malware detection through the development of adversarial malware images and the retraining of classifiers for increased robustness [33]. The approach is not, however, reliant mainly on adversarial training, which can be computationally costly and cannot, by definition, generalize to new types of attacks [34].
autonomous penetration testing mechanism through Conditional GANs for the creation of attack payloads (such as XSS) for web applications, particularly for WSN and IoT applications [35]. The model's dependence on semantic tokenization and training data may, however, restrict its generalizability across varied real-world applications and novel attack patterns [36]. Uncertainty-based adversarial test set generation technique to make autonomous driving systems more resistant against data perturbations and adversarial attacks [37]. However, the limitation of their approach is that it depends on model retraining, which would not generalise well to other driving situations or undiscovered attack forms [38].

To improve the efficacy and interpretability of AI model evaluation, propose a similarity-based adversarial testing strategy for CNNs, in which attack target labels are selected based on semantic and network-based similarity [39]. However, the method's reliance on similarity metrics may limit its use with different datasets or model architectures that are more difficult to interpret [40]. The increasing danger of hostile AI in cyber security, mentioning how AI-facilitated attacks such as evasion, poisoning, and adversarial malware are capable of evading even sophisticated defence mechanisms [41]. The limitation of the study, however, lies in its reliance on survey-driven information instead of proving actual implementation or performance of the suggested defence mechanisms [42].

Integrated framework that unites Adversarial Machine Learning (AML) and AI-powered data engineering for real-time adaptive cybersecurity and fraud detection [43]. The reliance of the method on continuous data evolution and system adaptability could, however, restrict its effectiveness in highly dynamic or new attack scenarios [44]. Defence mechanism through a modified ResNet implemented on edge clouds to safeguard metaverse AI applications against adversarial attacks with high success rates on typical datasets [45]. The method, however, depends significantly on large-scale training with varied data and does not test under real-time latency-constrained metaverse scenarios [46].

Artificial intelligence system that enhances CNN classifiers' resilience in noisy production settings by combining adversarial training and defensive methods [47]. However, the strategy's reliance on hybrid defence mechanisms may limit its scalability and effectiveness for real-time industrial use [48].AI-driven system for elderly care by combining IoT sensors and Generative Adversarial Networks to create synthetic healthcare data for customized monitoring in smart homes [49]. However, the approach may have issues with retaining consistently high-quality GAN-generated data across a variety of medical situations and scaling for real-time deployment [50].

Computer vision system protections and adversarial evasion threats, offering a workable approach for assessing machine learning models in safety-critical use cases like healthcare and driverless cars [51]. However, their findings show that even very advanced models like ResNet frequently fail to meet safety-critical requirements, highlighting significant flaws in current protections [52]. The issue of limited fault samples in rotor-bearing systems and enhance AI-based condition monitoring, present a hybrid fault classification approach based on FEM simulations and GANs [53]. However, the suggested approach might not be able to handle the noise and real-time fault variation that are unavoidable in actual industrial settings [54].

The study on computer vision system protections and adversarial evasion threats presents a crucial evaluation of machine learning models, particularly in safety-critical applications such as healthcare and autonomous vehicles [55]. While the research offers valuable insights into how machine learning models can be assessed for vulnerabilities in these high-stakes environments, it underscores a significant issue: even advanced architectures like ResNet often fail to meet the stringent safety and reliability standards required for such applications [56]. The findings highlight that despite their sophistication; current protections are insufficient to prevent adversarial attacks that can cause severe consequences in real-world use cases [57]. This exposes a major flaw in the design and deployment of machine learning systems in critical fields, suggesting that further advancements are needed to enhance robustness and ensure the models can withstand malicious or unexpected inputs without compromising safety [58].

In the context of industrial AI applications, the study also addresses the challenge of fault detection in rotor-bearing systems, particularly with the limited availability of fault samples [59]. To enhance AI-based condition monitoring, a hybrid fault classification approach was proposed, combining Finite Element Method (FEM) simulations with Generative Adversarial Networks (GANs) [60]. While the proposed approach holds promise for improving fault detection and prediction in industrial settings, it faces significant limitations [61] when confronted with the realities of noisy data and the dynamic nature of fault conditions in real-time operations [62]. In practice, the noise inherent in sensor data and the constant variability of fault conditions in industrial environments make it difficult for the model to maintain high accuracy and reliability [63]. As such, the hybrid approach may struggle to deliver the consistency needed for effective monitoring in complex, real-world scenarios, highlighting the need for further refinement and adaptation to cope with the challenges of noise and unpredictable fault variations [64].

3. Problem Statement

Even with recent breakthroughs in AI-based adversarial test-case generation software, a number of fundamental limitations hold back the scalability, efficiency, and applicability of current solutions [65]. In the first place, most generative [66],[67].AI methods in test-case generation are plagued by high computational complexity and reliance on high-quality, domain-specific data, which restricts their generalizability and scalability across varied real-world settings[68], Second, adversarial training methods, although highly effective at improving model resiliencegenerally [69]. do not generalize to new attack channels and involve massive computational [70],expense at retraining time Third, combined or composite defence approaches involving the integration of multiple AI mechanisms will compromise real-time performance and scalability [71] especially in production settings where fast feedback and low latency are essential [72] Finally, overdependence on similarity measures and semantic feature representations in test generation in an adversarial context [73]may compromise the interpretability and portability of test cases across varied software architectures and datasets [74] To overcome these difficulties, this paper suggests,[75] a light-weight adversarial testing method that uses Convolutional Neural Networks[76] (CNN) to learn software behaviour and detect vulnerable input areas, along with Particle Swarm Optimization[77] (PSO) for generating efficient adversarial test cases [78]. The overall strategy uses this combined framework to optimize [79]. fault detection efficiency, minimize test generation time, and reduce computational overhead [80]—suited for scalable, real-time software testing in dynamic environments [81].

4. Proposed Methodology

This picture depicts a system in which a CNN is trained to identify patterns in software defects, and a PSO algorithm generates adversarial inputs to confirm the model's resilience. Metrics including accuracy, F1 score, assault success, and robustness are used to test the system.

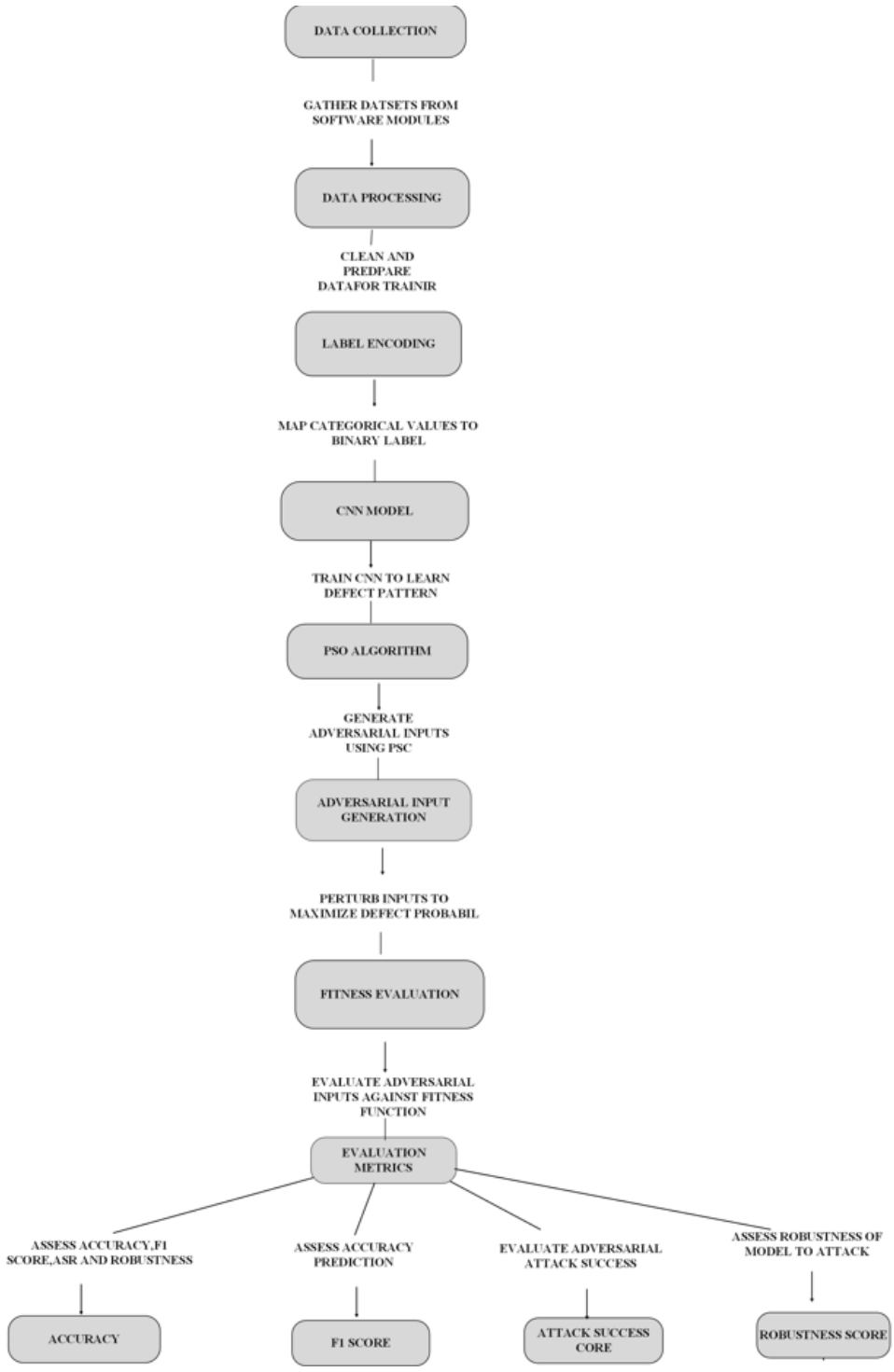

Figure 1: Overall Methodology Flow Diagram

4.1 Data collection

Software Defect Prediction Metrics from code modules that are classes or files make up the dataset, which is collected from real software projects and mostly acquired from sources like the NASA and PROMISE datasets. Lines of code (LOC), cyclomatic complexity, Halstead metrics, and object-oriented design metrics are among the static code qualities that make up these metrics, which are used to determine whether or not a module is prone to defects. In the software development lifecycle, automated static analysis tools are typically used to collect the data. This enables researchers to build, train, and test models in order to find software flaws before they are implemented.

4.2 Data Preprocessing

During the data preprocessing phase, a few crucial steps are implemented to keep the dataset clean and prepared

for training the model. The missing value removal is first done by removing rows with null or missing values to keep the input data intact. This makes sure that the model will not be impacted by incomplete records. For the target variable, label encoding is used to map the categorical variable (for example, defective or not defective) to binary labels (0 for non-defective and 1 for defective) so that it can be used for machine learning models.

4.2.1. Missing value Removal

Missing value removal is the process of eliminating rows (or sometimes columns) from the dataset that contain null, NaN, or undefined values.

To clean the dataset DDD by removing any rows that contain missing values formula shown in Equation (1):

$$D_{\text{clean}} = \{x \in D \mid \forall f \in x, f \neq \text{NaN}\} \quad (1)$$

4.2.2 Label Encoding

Label encoding is the process of changing categorical text labels (like "defective" or "non-defective") into numbers for machine learning algorithms. In binary classification, every category is assigned a distinct number, for example, 0 for "non-defective" and 1 for "defective." This enables models to operate mathematically on the labels while being trained.

Convert the categorical target variable y into binary labels suitable for machine learning formula shown in Equation (2):

$$y_{\text{encoded}} = \begin{cases} 1, & \text{if } y = \text{" defective"} \\ 0, & \text{if } y = \text{" non-defective"} \end{cases} \quad (2)$$

4.2.3 Final Training Dataset Construction

Final training dataset build is the combination of the cleaned data and the transformed labels (such as 0 or 1) into a single complete dataset. Every row contains input values (features) and the correct solution (label). The model is trained on this entire dataset.

Utilising the clean feature data and encoded labels formula found in Equation, create the final training dataset. shown in Equation (3):

$$T = \{(x, y_{\text{encoded}}) \mid x \in X_{\text{clean}}, y_{\text{encoded}} \in \{0,1\}\} \quad (3)$$

4.3 Algorithm Descriptions

The suggested framework combines a light-weight Convolutional Neural Network (CNN) with Particle Swarm Optimization (PSO) to produce adversarial inputs for software fault prediction. The hybrid model seeks to effectively detect severe vulnerabilities by learning from static software characteristics and searching the input space for fault-prone areas.

4.3.1 CNN Model

The CNN component is in charge of figuring out the fundamental trends connected to software flaws. Lines of Code (LOC), cyclomatic complexity, Halstead measures, and object-oriented metrics are among the code metrics that are vectorised and fed into it. The model produces a probability score that indicates the likelihood of the software module being defective. It is made up of two 1D convolutional layers and a fully linked dense layer.

Let $x_i \in \mathbb{R}^n$ be the input feature vector for the i^{th} software module, and let $\hat{y}_i = f(x_i; \theta)$ represent the CNN model's output, where θ denotes the learnable parameters of the CNN. The CNN extracts high-level feature representations through a sequence of transformations. The equation for the convolution operation formula shown in Equation (4):

$$z = \sigma(W * x + b) \quad (4)$$

where σ is the non-linear activation function, b is the bias term, * indicates the convolution operation, and W is the convolutional kernel. In hidden layers, the activation function is ReLU formula shown in Equation (5):

$$\sigma(z) = \max(0, z) \quad (5)$$

The output layer uses a sigmoid activation to perform binary classification between defective and non-defective modules formula shown in Equation (6):

$$\hat{y}_i = \frac{1}{1+e^{-z}} \quad (6)$$

This predicted probability score is then used as a basis for evaluating adversarial inputs generated via PSO.

4.3.2 Particle Swarm Optimization (PSO)

To provide the adversarial test inputs that maximize the defect probability predicted by the CNN. Each particle in the swarm represents a potential test input vector $x_i' \in \mathbb{R}^n$, initialized by perturbing valid input samples within realistic and bounded feature ranges.

The particles adjust their positions based on both their individual best and the global best positions discovered thus far, according to the following update rules formula, which is displayed in Equation (7):

$$v_i^{(t+1)} = w \cdot v_i^{(t)} + c_1 \cdot r_1 \cdot \left(p_{\text{best}}^{(i)} - x_i^{(t)}\right) + c_2 \cdot r_2 \cdot \left(g_{\text{best}} - x_i^{(t)}\right)$$
$$x_i^{(t+1)} = x_i^{(t)} + v_i^{(t+1)} \quad (7)$$

where w is the inertia weight regulating exploration vs. exploitation, v_i is the particle I's velocity vector, and x_i is its current position (input vector). C_1, C_2 are the coefficients of cognitive and social acceleration, r_1, $r\_2 \sim U(0,1)$ are random variables, $p\_"best"\ "\ \hat{}\ ((i))$ is the particle's optimal location, and $g\_"best"\ "\ is$ the swarm's optimal global position.

4.3.3 Fitness Function

To guide the optimization process, a fitness function is defined to evaluate each particle (test case) based on its potential to expose software vulnerabilities. The fitness score $F(x_i)$ for a candidate input vector x_i is computed as a weighted combination of three key components formula shown in Equation (8):

$$F(x_i) = \alpha \cdot \hat{y}_i + \beta \cdot C(x_i) + \gamma \cdot E(x_i) \quad (8)$$

Where, $y\hat{}\_i = f(x\_i; \theta)$ is the predicted defect probability from the CNN, $C(x_i)$ is the code coverage achieved when executing x_i, $E(x_i)$ is a binary indicator for whether the input causes a crash, assertion failure, or exception, α, β, γ are tunable weights to balance the components.

The CNN prediction component $\hat{y}_i$ encourages generation of inputs that are likely to be defective, while the code coverage term $C(x_i)$ promotes exploration of new paths in the software. The error indicator $E(x_i)$ prioritizes test cases that cause actual software failures.

4.4 Architecture Model

The basic architecture of a convolutional neural network (CNN) is depicted in this figure. The input layer is where it begins, receiving data (such a picture). Following the identification of important characteristics by the convolutional layers, the data is subsequently reduced in size by the pooling layers. Following feature identification, the fully connected layers process the data to generate predictions, like picture classification. Together, these various elements enable the model to recognise patterns and generate accurate predictions.

Let $f(x; \theta)$ be the CNN model with the parameters θ predicting the output $y\hat{}\_i = f(x\_i; \theta)$. The objective is to pass an input sample x_i through the CNN architecture in order to output a predicted value y ˆ_i, potentially a probability in this binary case. The CNN employs several key operations in its architecture:

4.4.1 1D Convolution

Convolution operation is the fundamental operation in CNNs and is utilized for feature extraction from the input data. A 1D convolution is performed over the input signal, e.g., time-series data or sequence, with filters (or kernels). Convolution operation takes a filter W and slides it over a part of the input x, computing the weighted sum at every position. Mathematically, the 1D convolution operation formula shown in Equation (9):

$$z = \sigma(W * x + b) \quad (9)$$

In this case, W stands for the convolution filter (weights), x for the input data, b for the bias term, which indicates the convolution process, and σ for the activation function (applied after that).

4.4.2. ReLU Activation

ReLU (Rectified Linear Unit) is also the activation functions within CNNs commonly applied for inducing non-linearity in the model. ReLU allows the model to learn detailed patterns since it allows it to activate only the positive part of the convolution's output and cuts off negative inputs. The ReLU function formula shown in Equation (10):

$$\sigma(z) = \max(0, z) \quad (10)$$

Where, z is the input to the activation function (the result of the convolution), σ(z) is the output after the ReLU activation.

4.4.3. Sigmoid Output for Binary Classification

For binary classification tasks (e.g., software system defect prediction), the CNN model will generally employ a Sigmoid activation function on the output layer so that it can output values between 0 and 1. This is appropriately suited to models that produce binary classes as predictions (e.g., non-defective or defective). The Sigmoid function formula shown in Equation (11):

$$\hat{y}_i = \frac{1}{1+e^{-z}} \quad (11)$$

where $y\hat{}\_i$ is the predicted probability for class 1 (defective), $e\hat{}(-z)$ is the exponential function, and z is the output from the network's final layer (following the application of the convolution and ReLU layers).

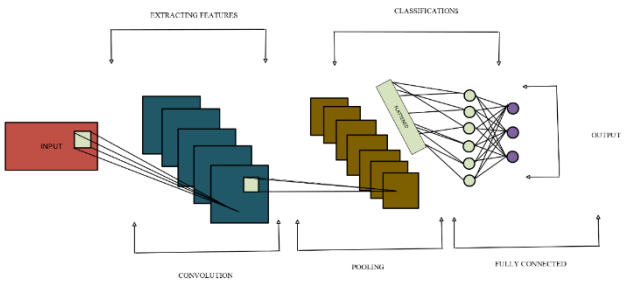

Figure 2: CNN Architecture for Feature Extraction and Classification

5. Result

The results section compares the performance of the suggested lightweight adversarial software testing framework utilizing Convolutional Neural Networks (CNN) and Particle Swarm Optimization (PSO). Lines of Code (LOC), cyclomatic complexity, Halstead measures, and object-oriented design measurements are among the static code measures that are experimentally compared on the Software Defect Prediction dataset. It is aimed to investigate how effectively the suggested framework will identify defects within software systems using PSO generated adversarial test cases based on CNN's output.

5.1 Model Performance Metrics

Model performance is measured based on some of the conventional evaluation measures, which include accuracy, F1 score, Attack Success Rate (ASR), and robustness score. They facilitate identification of how good or bad the generation of adversarial test cases and prediction by the model towards the detection of faulty and clean software modules are,

5.1.1 Accuracy

Accuracy is a basic performance measurement that evaluates the general performance of a model. The proportion of accurate predictions both true positives and true negatives—to all of the model's predictions is known as accuracy. The formula used to determine accuracy formula shown in Equation (13):

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN} \quad (13)$$

5.1.2 F1 score

The F1 Score is a statistic for evaluation that strikes a balance between recall and precision. When working with class-imbalanced datasets—datasets in which one class is significantly more prevalent than the other—it is quite helpful. Since the F1 score accounts for both false positives and false negatives, it provides a more accurate picture in these circumstances, while accuracy can be highly misleading.

The F1 score formula shown in Equation (14):

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \quad (15)$$

5.1.3 Attack success rate

Attack Success Rate (ASR) is a key measure to assess the efficiency of adversarial attacks on a model. ASR quantifies how frequently an adversarial attack is able to make the model produce an incorrect prediction (misclassification). The measure is notably useful in the context of adversarial testing, where one desires to assess with which ease a model can be deceived by perturbations in the input that are small but significant (adversarial inputs). The attack success formula shown in equation (15):

$$ASR = \frac{\text{Number of Successful Adversarial Attacks}}{\text{Total Adversarial Samples}} \quad (15)$$

5.2 Summary of Results

The proposed hybrid framework combining CNN and PSO outperforms individual CNN and PSO models across all key performance metrics. Below is a summary table of the results:

Table 1: Performance Metrics of the Methodology

Model	Accuracy (%)	F1 Score (%)	ASR (%)	Robustness (%)	Faults Detected (%)	Time (Seconds)
Proposed Framework	**99.35**	**99.35**	**3.45**	**96.55**	**99.60**	**10.5**

This table makes it evident that the suggested framework is superior in terms of efficiency and performance, which makes it appropriate for situations involving extensive, real-time software testing. This bar graph displays the Proposed Framework's Attack Success Rate (ASR). It makes it easier to compare how well the model performs in the face of hostile attacks.

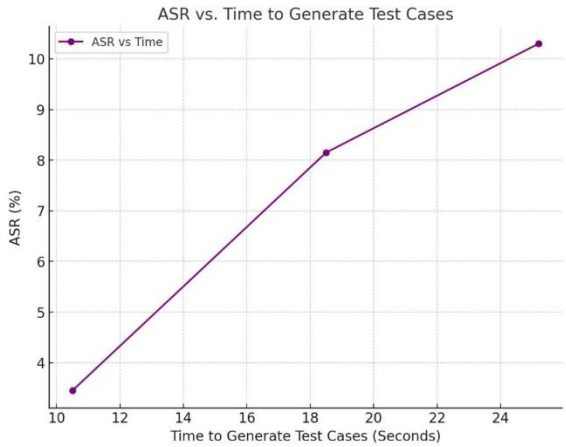

Figure 3: ASR VS Time to Generate Teat Cases

The "ASR vs. Time to Generate Test Cases" line graph illustrates how the Attack Success Rate (ASR) varies with the amount of time needed to create adversarial test cases. The graph indicates a positive relationship, where ASR increases when the time to generate test cases rises. This implies that the success rate of adversarial attacks increases as more time is spent on test case generation, and it may be the case that longer test generation times reveal more vulnerabilities or weaknesses in the system. This is a trend which provides insight into the compromise between how much time is spent on test case generation and how well the model is attacked by adversarial attacks.

This is the histogram representing the distribution of Faults Detected (%) across various time periods. The histogram serves to represent how often specific values for faults are detected across simulated time periods. The histogram "Distribution of Faults Detected (%) Over Time Intervals" graphically displays the distribution of Faults Detected (%) in different time intervals. The distribution seems to peak around 99.6%, with a majority of faults being detected in this range, pointing towards the majority of the test cases lying within a small interval of

almost fault-free detection. The shape of the histogram indicates that the system is working at its best, identifying faults consistently over time, with some fluctuations observed at the upper and lower ends. This indicates the model's capability to identify faults with high accuracy, as well as investigating various intervals in fault detection efficiency.

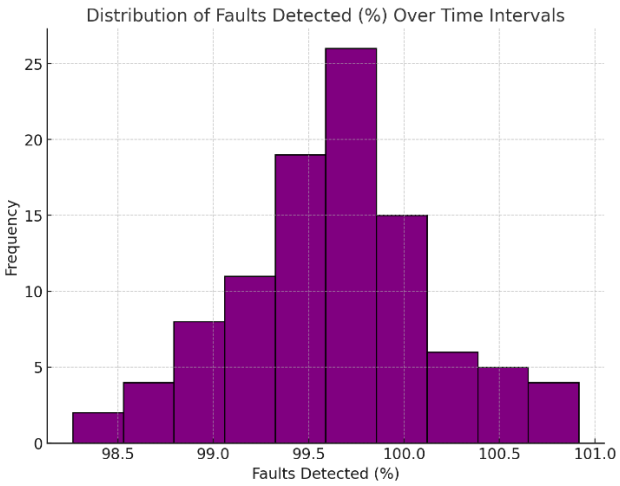

Figure 4: Distribution of Faults Detected Over Time Intervals

Table 2: Performance Comparison of the proposed Methodology

Model	Accuracy	F1 Score	Attack Success Rate (ASR)	Robustness	Faults Detected	Time to Generate Test Cases
Proposed Framework (CNN + PSO)	High	High	Low	High	High	Low
Generative AI Model [24]	Moderate	Moderate	Moderate	Moderate	Moderate	Moderate
Variational Auto-encoder Generative Adversarial Network[25]	Moderate	Moderate	High	Moderate	Moderate	High
Deep Variational Anomaly Generation[26]	Low	Low	High	Low	Low	High

Conclusion and Future Works

The main objectives of this research to develop a lightweight and low-complexity framework for adversarial software testing. To compare the model's performance in terms of robustness, attack success rate, F1 score, and accuracy. Research problem resolved by this study is the computational cost of the standard adversarial test methods on state-of-the-art optimization approaches or gradient-based systems. Future research will include combining multi-objective optimization, explainable AI, and adaptive defences mechanisms to improve model robustness. It can be improved through the application of multi-objective PSO, inclusion of explainable AI for transparency in the model, and adaptive defences.

References

[1] Zulkifley, M. A., Abdani, S. R., & Zulkifley, N. H. (2020). COVID-19 screening using a lightweight convolutional neural network with generative adversarial network data augmentation. Symmetry, 12(9), 1530.

[2] Pulakhandam, W., & Samudrala, V. K. (2020). Automated Threat Intelligence Integration To Strengthen SHACS For Robust Security In Cloud-Based Healthcare Applications. International Journal of Engineering & Science Research, 10(4).

[3] Liu, X., Du, X., Zhang, X., Zhu, Q., Wang, H., & Guizani, M. (2019). Adversarial samples on android malware detection systems for IoT systems. Sensors, 19(4), 974.

[4] Dondapati, K. (2020). Clinical implications of big data in predicting cardiovascular disease using SMOTE for handling imbalanced data. Journal of Cardiovascular Disease Research, 11(9), 191-202.

[5] Sahoo, D. P., Nguyen, P. H., Mukhopadhyay, D., & Chakraborty, R. S. (2015). A case of lightweight PUF constructions: Cryptanalysis and machine learning attacks. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 34(8), 1334-1343.

[6] Grandhi, S. H. (2020). Blockchain-enabled software development traceability: Ensuring secure and transparent software lifecycle management. International Journal of Information Technology & Computer Engineering, 8(3)

[7] Sagar, R., Jhaveri, R., & Borrego, C. (2020). Applications in security and evasions in machine learning: a survey. Electronics, 9(1), 97.

[8] Natarajan, D. R. (2020). AI-Generated Test Automation for Autonomous Software Verification: Enhancing Quality Assurance Through AI-Driven Testing. Journal of Science and Technology, 5(5).

[9] Park, H., & Baek, N. (2020). Developing an open-source lightweight game engine with DNN support. Electronics, 9(9), 1421.

[10] Srinivasan, K. (2020). Neural network-driven Bayesian trust prediction model for dynamic resource management in cloud computing and big data. International Journal of Applied Science Engineering and Management, 14(1).

[11] Xiao, J., Wu, H., & Li, X. (2019). Internet of things meets vehicles: Sheltering in-vehicle network through lightweight machine learning. Symmetry, 11(11), 1388.

[12] Chauhan, G. S. (2020).Utilizing data mining and neural networks to optimize clinical decision-making and patient

outcome predictions. International Journal of Marketing Management, 8(4), 32-51.
[13] Wu, F., Yang, W., Xiao, L., & Zhu, J. (2020). Adaptive wiener filter and natural noise to eliminate adversarial perturbation. Electronics, 9(10), 1634.
[14] Gollapalli, V. S. T. (2020). Enhancing disease strati fication using federated learning and big data analytics in healthcare systems. International Journal of Management Research and Business Strategy, 10(4), 19-38.
[15] Wang, G., & Ren, P. (2020). Hyperspectral image classification with feature-oriented adversarial active learning. Remote Sensing, 12(23), 3879.
[16] Gollapalli, V. S. T. (2020). Scalable Healthcare Analytics in the Cloud: Applying Bayesian Networks, Genetic Algorithms, and LightGBM for Pediatric Readmission Forecasting. International Journal of Life Sciences Biotechnology Pharma Sciences, 16(2).
[17] Salah, A., Shalabi, E., & Khedr, W. (2020). A lightweight android malware classifier using novel feature selection methods. Symmetry, 12(5), 858.
[18] Ganesan, T. (2020). Deep learning and predictive analytics for personalized healthcare: unlocking ehr insights for patient-centric decision support and resource optimization. International Journal of HRM and Organizational Behavior, 8(3).
[19] Bock, J. R., & Maewal, A. (2020). Adversarial learning for product recommendation. Ai, 1(3), 25.
[20] Panga, N. K. R., & Thanjaivadivel, M. (2020). Adaptive DBSCAN and Federated Learning-Based Anomaly Detection for Resilient Intrusion Detection in Internet of Things Networks. International Journal of Management Research and Business Strategy, 10(4).
[21] Karn, R. R., Kudva, P., Huang, H., Suneja, S., & Elfadel, I. M. (2020). Cryptomining detection in container clouds using system calls and explainable machine learning. IEEE transactions on parallel and distributed systems, 32(3), 674-691.
[22] Dyavani, N. R., & Hemnath, R. (2020). Blockchain-integrated cloud software networks for secure and efficient ISP federation in large-scale networking environments. International Journal of Engineering Research and Science & Technology, 16(2). https://ijerst.org/index.php/ijerst/ article/view/614/558
[23] Soe, Y. N., Feng, Y., Santosa, P. I., Hartanto, R., & Sakurai, K. (2020). Towards a lightweight detection system for cyber attacks in the IoT environment using corresponding features. Electronics, 9(1), 144.
[24] Durai Rajesh Natarajan, & Sai Sathish Kethu. (2019). Decentralized anomaly detection in federated learning: Integrating one-class SVM, LSTM networks, and secure multi-party computation on Ethereum blockchain. International Journal of Computer Science Engineering Techniques, 5(4).
[25] Sayghe, A., Hu, Y., Zografopoulos, I., Liu, X., Dutta, R. G., Jin, Y., & Konstantinou, C. (2020). Survey of machine learning methods for detecting false data injection attacks in power systems. IET Smart Grid, 3(5), 581-595.
[26] Nagarajan, H., & Kurunthachalam, A. (2018). Optimizing database management for big data in cloud environments. International Journal of Modern Electronics and Communication Engineering, 6(1).
[27] Mukhopadhyay, D. (2016). PUFs as promising tools for security in internet of things. IEEE Design & Test, 33(3), 103-115.
[28] Basani, D. K. R., & Aiswarya, R. S. (2018). Integrating IoT and robotics for autonomous signal processing in smart environment. International Journal of Information Technology and Computer Engineering, 6(2).
[29] Rostami, M., Majzoobi, M., Koushanfar, F., Wallach, D. S., & Devadas, S. (2014). Robust and reverse-engineering resilient PUF authentication and key-exchange by substring matching. IEEE Transactions on Emerging Topics in Computing, 2(1), 37-49.
[30] Gudivaka, B. R., & Palanisamy, P. (2018). Enhancing software testing and defect prediction using Long Short-Term Memory, robotics, and cloud computing. International Journal of modern electronics and communication Engineering, 6(1).
[31] Wu, H., Li, X., & Deng, Y. (2020). Deep learning-driven wireless communication for edge-cloud computing: opportunities and challenges. Journal of Cloud Computing, 9(1), 21.
[32] Kodadi, S., & Kumar, V. (2018). Lightweight deep learning for efficient bug prediction in software development and cloud-based code analysis. International Journal of Information Technology and Computer Engineering, 6(1).
[33] Chen, Y., Xie, Y., Song, L., Chen, F., & Tang, T. (2020). A survey of accelerator architectures for deep neural networks. Engineering, 6(3), 264-274.
[34] Bobba, J., & Prema, R. (2018). Secure financial data management using Twofish encryption and cloud storage solutions. International Journal of Computer Science Engineering Techniques, 3(4), 10–16.
[35] Xue, M., Gu, C., Liu, W., Yu, S., & O'Neill, M. (2020). Ten years of hardware Trojans: a survey from the attacker's perspective. IET Computers & Digital Techniques, 14(6), 231-246.
[36] Gollavilli, V. S. B., & Thanjaivadivel, M. (2018). Cloud-enabled pedestrian safety and risk prediction in VANETs using hybrid CNN-LSTM models. International Journal of Information Technology and Computer Engineering, 6(4), 77–85. ISSN 2347–3657.
[37] Kang, J., & Gwak, J. (2020). Ensemble learning of lightweight deep learning models using knowledge distillation for image classification. Mathematics, 8(10), 1652.
[38] Nippatla, R. P., & Palanisamy, P. (2018). Enhancing cloud computing with eBPF powered SDN for secure and scalable network virtualization. Indo-American Journal of Life Sciences and Biotechnology, 15(2).
[39] Fernando, D. W., Komninos, N., & Chen, T. (2020). A study on the evolution of ransomware detection using machine learning and deep learning techniques. IoT, 1(2), 551-604.
[40] Budda, R., & Pushpakumar, R. (2018). Cloud Computing in Healthcare for Enhancing Patient Care and Efficiency. Chinese Traditional Medicine Journal, 1(3), 10-15.
[41] Dang, L. M., Hassan, S. I., Im, S., Lee, J., Lee, S., & Moon, H. (2018). Deep learning based computer generated face identification using convolutional neural network. Applied Sciences, 8(12), 2610.
[42] Vallu, V. R., & Palanisamy, P. (2018). AI-driven liver cancer diagnosis and treatment using cloud computing in healthcare. Indo-American Journal of Life Sciences and Biotechnology, 15(1).
[43] Zhuang, H., Xi, X., Sun, N., & Orshansky, M. (2019). A strong subthreshold current array PUF resilient to machine learning attacks. IEEE Transactions on Circuits and Systems I: Regular Papers, 67(1), 135-144.
[44] Jayaprakasam, B. S., & Hemnath, R. (2018). Optimized microgrid energy management with cloud-based data analytics and predictive modelling. International Journal of modern electronics and communication Engineering, 6(3), 79–87.
[45] Gopalakrishnan, K. (2018). Deep learning in data-driven pavement image analysis and automated distress detection: A review. Data, 3(3), 28.
[46] Mandala, R. R., & N, Purandhar. (2018). Optimizing secure cloud-enabled telemedicine system using LSTM with stochastic gradient descent. Journal of Science and Technology, 3(2).
[47] Cem Birbiri, U., Hamidinekoo, A., Grall, A., Malcolm, P., & Zwiggelaar, R. (2020). Investigating the performance of generative adversarial networks for prostate tissue detection and segmentation. Journal of imaging, 6(9), 83.
[48] Garikipati, V., & Palanisamy, P. (2018). Quantum-resistant cyber defence in nation-state warfare: Mitigating threats with

post-quantum cryptography. Indo-American Journal of Life Sciences and Biotechnology, 15(3).

[49] Paredes, J. N., Simari, G. I., Martinez, M. V., & Falappa, M. A. (2018). First steps towards data-driven adversarial deduplication. Information, 9(8), 189.

[50] Ubagaram, C., & Mekala, R. (2018). Enhancing data privacy in cloud computing with blockchain: A secure and decentralized approach. International Journal of Engineering & Science Research, 8(3), 226–233.

[51] Cui, Y., Gu, C., Ma, Q., Fang, Y., Wang, C., O'Neill, M., & Liu, W. (2020). Lightweight modeling attack-resistant multiplexer-based multi-PUF (MMPUF) design on FPGA. Electronics, 9(5), 815.

[52] Ganesan, S., & Kurunthachalam, A. (2018). Enhancing financial predictions using LSTM and cloud technologies: A data-driven approach. Indo-American Journal of Life Sciences and Biotechnology, 15(1).

[53] Gao, H., Chen, Z., Huang, B., Chen, J., & Li, Z. (2020). Image super-resolution based on conditional generative adversarial network. IET Image Processing, 14(13), 3006-3013.

[54] Musam, V. S., & Kumar, V. (2018). Cloud-enabled federated learning with graph neural networks for privacy-preserving financial fraud detection. Journal of Science and Technology, 3(1).

[55] Muhammad, K., Ullah, A., Lloret, J., Del Ser, J., & De Albuquerque, V. H. C. (2020). Deep learning for safe autonomous driving: Current challenges and future directions. IEEE Transactions on Intelligent Transportation Systems, 22(7), 4316-4336.

[56] Musham, N. K., & Pushpakumar, R. (2018). Securing cloud infrastructure in banking using encryption-driven strategies for data protection and compliance. International Journal of Computer Science Engineering Techniques, 3(5), 33–39.

[57] Asharf, J., Moustafa, N., Khurshid, H., Debie, E., Haider, W., & Wahab, A. (2020). A review of intrusion detection systems using machine and deep learning in internet of things: Challenges, solutions and future directions. Electronics, 9(7), 1177.

[58] Radhakrishnan, P., & Mekala, R. (2018). AI-Powered Cloud Commerce: Enhancing Personalization and Dynamic Pricing Strategies. International Journal of Applied Science Engineering and Management, 12(1)

[59] Butt, U. A., Mehmood, M., Shah, S. B. H., Amin, R., Shaukat, M. W., Raza, S. M., ... & Piran, M. J. (2020). A review of machine learning algorithms for cloud computing security. Electronics, 9(9), 1379.

[60] Nagarajan, H., & Kumar, R. L. (2020). Enhancing healthcare data integrity and security through blockchain and cloud computing integration solutions. International Journal of Engineering Technology Research & Management, 4(2).

[61] Kwon, D., Malaiya, R., Yoon, G., Ryu, J. T., & Pi, S. Y. (2019). A study on development of the camera-based blind spot detection system using the deep learning methodology. Applied Sciences, 9(14), 2941.

[62] Gudivaka, B. R., & Thanjaivadivel, M. (2020). IoT-driven signal processing for enhanced robotic navigation systems. International Journal of Engineering Technology Research & Management, 4(5).

[63] Chen, J., & Ran, X. (2019). Deep learning with edge computing: A review. Proceedings of the IEEE, 107(8), 1655-1674.

[64] Chetlapalli, H., & Pushpakumar, R. (2020). Enhancing accuracy and efficiency in AI-driven software defect prediction automation. International Journal of Engineering Technology Research & Management, 4(8).

[65] Chen, C., Qin, C., Qiu, H., Tarroni, G., Duan, J., Bai, W., & Rueckert, D. (2020). Deep learning for cardiac image segmentation: a review. Frontiers in cardiovascular medicine, 7, 25.

[66] Budda, R., & Mekala, R. (2020). Cloud-enabled medical image analysis using ResNet-101 and optimized adaptive moment estimation with weight decay optimization. International Research Journal of Education and Technology, 03(02).

[67] Gu, P., Zhu, C., Lan, X., Wang, J., & Li, S. (2020). Robust Image Classification with Cognitive-Driven Color Priors. Electronics, 9(11), 1837.

[68] Vallu, V. R., & Rathna, S. (2020). Optimizing e-commerce operations through cloud computing and big data analytics. International Research Journal of Education and Technology, 03(06).

[69] Xiao, C., Han, D., Ma, Y., & Qin, Z. (2019). CsiGAN: Robust channel state information-based activity recognition with GANs. IEEE Internet of Things Journal, 6(6), 10191-10204.

[70] Jayaprakasam, B. S., & Padmavathy, R. (2020). Autoencoder-based cloud framework for digital banking: A deep learning approach to fraud detection, risk analysis, and data security. International Research Journal of Education and Technology, 03(12).

[71] Chehab, M., & Mourad, A. (2020). LP-SBA-XACML: Lightweight semantics based scheme enabling intelligent behavior-aware privacy for IoT. IEEE Transactions on Dependable and Secure Computing, 19(1), 161-175.

[72] Mandala, R. R., & Kumar, V. K. R. (2020). AI-driven health insurance prediction using graph neural networks and cloud integration. International Research Journal of Education and Technology, 03(10).

[73] Xu, H., Zhou, Y., Ming, J., & Lyu, M. (2020). Layered obfuscation: a taxonomy of software obfuscation techniques for layered security. Cybersecurity, 3, 1-18.

[74] Ubagaram, C., & Kurunthachalam, A. (2020). Bayesian-enhanced LSTM-GRU hybrid model for cloud-based stroke detection and early intervention. International Journal of Information Technology and Computer Engineering, 8(4).

[75] Murthy, C. B., Hashmi, M. F., Bokde, N. D., & Geem, Z. W. (2020). Investigations of object detection in images/videos using various deep learning techniques and embedded platforms—A comprehensive review. Applied sciences, 10(9), 3280.

[76] Ganesan, S., & Hemnath, R. (2020). Blockchain-enhanced cloud and big data systems for trustworthy clinical decision-making. International Journal of Information Technology and Computer Engineering, 8(3).

[77] Kashyap, P., Aydin, F., Potluri, S., Franzon, P. D., & Aysu, A. (2020). 2deep: Enhancing side-channel attacks on lattice-based key-exchange via 2-d deep learning. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 40(6), 1217-1229.

[78] Musam, V. S., & Purandhar, N. (2020). Enhancing agile software testing: A hybrid approach with TDD and AI-driven self-healing tests. International Journal of Information Technology and Computer Engineering, 8(2).

[79] Ullah, S., & Kim, D. H. (2020). Lightweight driver behavior identification model with sparse learning on in-vehicle can-bus sensor data. Sensors, 20(18), 5030.

[80] Musham, N. K., & Bharathidasan, S. (2020). Lightweight deep learning for efficient test case prioritization in software testing using MobileNet & TinyBERT. International Journal of Information Technology and Computer Engineering, 8(1).

[81] Sadeghi, A., Bagheri, H., Garcia, J., & Malek, S. (2016). A taxonomy and qualitative comparison of program analysis techniques for security assessment of android software. *IEEE Transactions on Software Engineering*, *43*(6), 492-530.