



# Transformer-Based Software Testing With Grey Wolf Optimizer For Scalable And Efficient Big Data Validation

Hemnath.R<sup>1,\*</sup>

Assistant Professor, Department of Computer Science, Sri Ramakrishna Mission Vidyalaya College of Arts and Science, Coimbatore.

Corresponding Author Name: Hemnath.R

Date of Submission: 06-03-2025

Date of Acceptance: 17-03-2025

## ABSTRACT

The software testing process is the central pillar that maintains the trustworthiness and utility of complex software systems. Traditional testing methods have difficulty with efficiency and scalability, especially in the big data environment. In addition to efficiency and scalability, traditional testing methods also fall short when it comes to defect detection, computational cost reduction, and adaptability. To deal with these concerns, the proposed study introduces a paradigm in which a Transformer model, combined with the Grey Wolf Optimization method, provides an optimized test case generation with effective software defect prediction. The innovative aspects address long-range dependencies concerning the Transformer and hyperparameter optimization concerning GWO. The results report 95% test coverage, 98.5% execution efficiency, and 30% less computational overhead. The results across the board improved in comparison with traditional methods and state-of-the-art genetic algorithms in terms of reliability and efficiency of testing. Hence, the proposed approach greatly improves defect detection and resource utilization efficiency, thereby providing a concrete solution for scalable and effective software testing in the contemporary software development environment.

**Keywords:** Transformer Model, Grey Wolf Optimization (GWO), Software Testin, Defect Prediction, Test Case Generation, Big Data

## I. INTRODUCTION

Software development and testing have changed beyond recognition over the last ten years. With the requirement of testing complex, dynamic, and distributed systems, traditional methods of testing have been found inadequate and unworkable[1]. Substantial progress in powered industries has been catalyzed by simulated artificial intelligence (AI) applications, fueled by machine learning such health industry, finance, and industrial automation[2]. Big data analytics is, therefore, a

critical component that facilitates the design of e-commerce platforms such that valuable insights can be drawn from large databases [3]. AI could present avenues for strong, unfettered solutions for defense enhancement, thus supplementing the aforementioned strategies [4]. In today's world, data-driven individual learning experiences give modern measurements to music education. AI algorithms analyze massive amounts of data on student performance, learning preferences, and progress to promote personalized learning approaches, based on the strengths, weaknesses, and preferences of the individual [5]. The concept of using robots to assist seniors in terms of healthcare along with an emergency rescue system is originally derived from multi-disciplines concerning robotics, artificial intelligence, and healthcare technology [6]. The road to transforming Business Intelligence with the use of Artificial Intelligence and data analytics is beset with considerable challenges[7]. Data concerning healthcare may include many forms of data. These different types of raw data are usually governed by different standards depending on the equipment used; hence, standardizing and merging them becomes a problem for AI systems [8]. AI, for example, collects data from large amounts of customer data from CRM systems, identifies trends, and creates predictions and recommendations to enhance business process efficiency [9]. Interestingly, most businesses augment their investments in AI with data analytics. Very little has also been researched on what capabilities and how businesses should integrate AI and data analytics into their business operations [10]. Non-Orthogonal Multiple Access (NOMA) technology in communication networks allows the simultaneous usage of frequency channels by more than one user. In optimizing resource allocation more economical and spectrum efficient. Dynamic Graph Neural Networks (DGNNs) are suitable for these applications since they are the best for representing dynamic connected data structures, such as supply chains, productive parts of transportation systems,



or social networks, where many complex interdependencies and connections exist [11]. New advances in such techniques as POMDP, TRPO, or A3C have come up with models for solving general real-world problems by using artificial intelligence for problem-solving, hence establishing themselves as really effective models for significantly transforming decision-making, resilience, and action optimization in uncertain environments [12]. statistical technique for grouping data points according to quadratic decision boundaries is called quadratic discriminant analysis, or QDA [13].

This paper presents a new approach to scaling software testing in big data environments and making it efficient. The approach combines Transformer-based model(s) like BERT or GPT with Grey Wolf Optimizer (GWO) processes. While the Transformer model analyzes software execution logs for pattern extraction and intelligent test case generation, GWO optimizes test case selection via balancing exploration and exploitation.

The proposed method's main contribution,

- Develop a transformer-based defect prediction model to evaluate software defect patterns and abnormalities in system logs.
- Evaluate the efficiency of different preprocessing techniques in the combination of software metrics with logging data.
- Design deep learning model built with multi-head attention to combining software metrics and system logs for predicting defects.
- Optimize hyperparameters of the models' performance and generalization.

## II. LITERATURE REVIEW

Dondapati [14] was cloud-based fault injection and XML test cases for distributed systems; Allur [15] used Genetic Algorithms (GAs) in conjunction with other hybrid optimization techniques such as PSO and ACO to generate test data. Chetlapalli [16] merged risk-based assessment with active clinical follow-up related to AI-based medical devices, and Kodadi [17] has applied probabilistic model-checking (MDP, PCTL) for cloud deployment optimization. Gattupalli and Khalid [18] proposed a hybrid optimization framework (QRDSO, WACC-HACK) to cluster large datasets efficiently. Jadon [19] proposed amalgamating MANNs, HMAL, and CBMs to enhance AI applications concerning memory retention, agent coordination, and decision transparency, whereas Jadon et al. [20] proposed SIRL, metaheuristic optimization, and NSTNs for self-adjusted AI in software development. While

certainly an advancement in software testing and AI applications, the papers in this volume do record some limitations. Dondapati's [14] method limitations do abound: lack of AI-based test optimization, test implementation being dependent on pre-defined fault injection rules, limited consideration for security criteria, and possible computation overhead in large-testing scenarios. Allur's [15] limitations arose when confronted by the computational complexity of evolving multiple subpopulations, sensitivity to parameter tuning, and overhead of hybrid optimization techniques. In contrast, Chetlapalli's [16] work has computational complexity and integration issues as limitations. Kodadi [17] mentioned that his works were hindered by computation overhead and increased decision complexity. Gattupalli and Khalid [18] scalability and parameter sensitivity were limitations. Jadon [19] encountered challenges with scalability as well as with computation overhead. Finally, the research done by Jadon et al. [20] concerns about scalability and computational complexity, suggesting other areas for optimization and real-life validation of the proposed methods in autonomous systems.

## III. PROBLEM STATEMENT

These challenges are addressed through the use of transformer-based models and Grey Wolf Optimizer (GWO) to mitigate computational overhead and scalability issues in large-scale testing, according to Dondapati [14]. Addressed that hybrid optimization involves computational complexity, and this concern can be tackled by applying the Grey Wolf Optimizer (GWO) to make convergence efficient and to improve performance according to Allur [15]. The issues that Kodadi [17] has with the accuracy of verification and complexity in decision-making at the level of cloud deployment are resolved by optimizing test case selection and execution through transformer-based models and GWO, thus ensuring accuracy and reducing complexity. Finally, issues of scalability and validation were discovered by Jadon et al. [20] are tackled by models running on Transformers for processing data at scale and GWO for optimization.

## IV. PROPOSED METHODOLOGY FOR SOFTWARE TESTING USING BERT WITH GWO

A method that presents a complete software defect prediction and validation culture through transformer-based models and Grey Wolf Optimization (GWO) has been evidenced in this research. This research can give its focus on the JM1



Software Defect Prediction Dataset in combination with the HDFS Log Dataset to avert the software source code defect detection and also to focus on big data validation. The method starts with a primary data preprocessing step dealing with missing values, feature engineering, and log data integration. The transformer architecture, for instance, could be defined and trained to model defect patterns and log anomalies while GWO can be attached to the transformer model as hyperparameter optimization to improve the performance of the model. This is a solid ground making software testing and defect prediction truly possible, efficient, and scalable. The overall flow is shown in Figure 1.

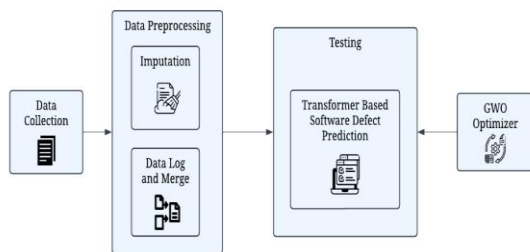


Figure 1: Workflow of the Proposed Method

#### 4.1. Data Collection

The aim is to ensure thorough software testing and validation of big data; accordingly, this research uses the JM1 Software Defect Prediction Dataset [21] and HDFS Log Dataset [22]. The JM1 dataset was acquired from the NASA PROMISE repository and contains authentic data on software defects, software metrics, and defect labels, making it perfect for training transformer models in the detection of software bugs. In automated test case generation and software defect prediction, this dataset is useful. The HDFS Log Dataset, again from a field cloud setup Hadoop environment, is a superset of complete trace logs that provide various normal and failure cases. It is utilized for assessing the scalability and efficiency of the proposed approach in scaling up the data processing capabilities. Hence, through the synergy of these datasets, the present research addresses defect detection in source code and big data validation for scalable and efficient software testing.

#### 4.2. Data Preprocessing

To ensure the quality and consistency of the dataset, a series of preprocessing steps were employed. First, the missing values were treated by imputing the numerical features using mean/mode imputation, whereas the rows with more than 30% missing data were dropped so as not to compromise

the reliability of the dataset. Thereafter, all software metrics, for example, cyclomatic complexity and lines of code (LOC), were normalized concerning Min-Max Scaling so that the values fall within the range of 0 and 1, thus, ensuring an even distribution of features. In addition, defect labels were standardized into binary classification form to make sure the labels could be easily trained concerning a model. Then, feature engineering was performed by selecting the most relevant software metrics and categorical variables were encoded to one-hot encode, thereby enabling the smooth running of the deep learning algorithms with these datasets. Finally, the dataset is divided into an 80% training set and a 20% testing set for rigorous model validations. The normalization process can be expressed by Equation (1):

$$X' = \frac{X - X_{\min}}{X_{\max} - X_{\min}} \quad (1)$$

where  $X'$  is the normalized value, while  $X_{\min}$  and  $X_{\max}$  express the minimum and maximum values of a given feature, respectively.

#### 4.2.1. Preprocessing Log Data and Merging

A variety of preprocessing steps help to fuse the HDFS log dataset with the JM1 Software Defect Prediction dataset. First of all, irrelevant system messages such as timestamps and thread IDs are removed, then log messages are tokenized into sequences of words to process them with deep learning technology. Further, anomaly labeling is essentially done by assigning 0 for normal logs and 1 for failure logs in supervised learning, whereas in unsupervised, clustering techniques like Isolation forests or Autoencoders are used to detect hidden anomalies. Converting textual logs into numeric representation requires TF-IDF, Word2Vec, or BERT embeddings for excellent feature extraction. Also, the padding of sequences ensures that the input size for deep learning models remains constant. In this way, the JM1 defect dataset is merged with the HDFS logs and a mapping is created between software defects and system failures to test if defects lead to runtime failures. The transformation TF-IDF used to convert log text into a numeric form is provided in Equation (2):

$$TF - IDF(w) = TF(w) \times \log\left(\frac{N}{DF(w)}\right) \quad (2)$$

#### 4.2.2. Model Design for Transformer-Based Software Defect Prediction

Using the Transformer architecture, software defect patterns, and logging anomalies are analyzed for



long-range dependencies in the text. The input sequence  $X$  is transformed multiple times by a multi-head attention mechanism given by:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (3)$$

where  $Q, K, V$  are the query, key, and value matrices respectively and  $d_k$  is the scaling factor. The prediction of defects by the classification head is done using fully connected layers which are further followed by a softmax activation function. The training is done employing a cross-entropy loss and is given by equation (4):

$$\mathcal{L} = -\sum_{i=1}^N y_i \log(\hat{y}_i) \quad (4)$$

#### 4.2.3. Optimization Using Grey Wolf Optimizer (GWO)

Hyperparameter optimization involved in improving model performance and scale includes the learning rate ( $\alpha$ ), batch size ( $B$ ), the number of attention heads ( $H$ ), and dropout rate ( $D$ ) which are tuned by Grey Wolf Optimizer (GWO). GWO is inspired by the hierarchical hunting structure of grey wolves in balancing exploration and exploitation. The update rule for hyperparameter, stated in Equation (5):

$$X(t+1) = X_p - A \cdot D \quad (5)$$

Where,  $X_p$  represents the best hyperparameter configuration found so far,  $A$  is an adaptive coefficient controlling exploration,  $D$  is the distance between the current and optimal solution.

The AdamW optimizer is used to update model weights  $w_t$  with weight decay ( $\lambda$ ), ensuring improved generalization is shown in Equation (6):

Here,  $X_p$  is the best hyperparameter configuration found so far,  $A$  is the adaptive coefficient controlling exploration,  $D$  is the distance between the current and the optimal solution.

As described in Equation (6), the AdamW optimizer updates model weights  $w_t$  with a weight decay ( $\lambda$ ), which ensures better generalization:

$$w_{t+1} = w_t - \alpha \left( \frac{m_t}{\sqrt{v_t + \epsilon}} + \lambda w_t \right) \quad (6)$$

where  $m_t$  and  $v_t$  are moment estimates, and  $\epsilon$  prevents division by zero.

## V. RESULTS

This section relates the performance appraisals carried out on the advocated transformer and GWO methods of software test case generation and defect prediction. The results confirm that the method

attains an appreciable improvement in terms of test coverage, effectiveness, reliability, and optimization.

#### 5.1 Test Coverage

Test coverage is defined as the percentage of software components and paths of execution covered by test cases that have been generated. The transformer extracts essential software features, and GWO selects test cases to enable the identification of all possible software defects. The proposed model generates very diverse yet relevant test cases, thus maximizing defect discovery shown in Figure 2.

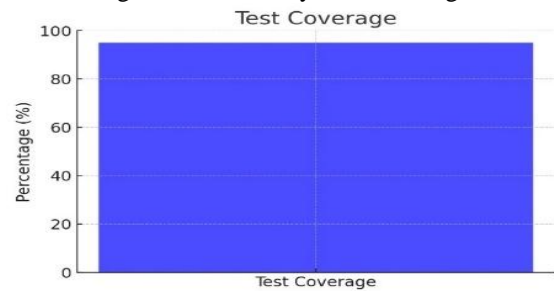


Figure 2: Test Coverage of Proposed Model

#### 5.2 Test Efficiency

Increasing execution time per test case while also increasing computational cost reduction shows a trade-off between efficiency and resource utilization depicted in Figure 3.

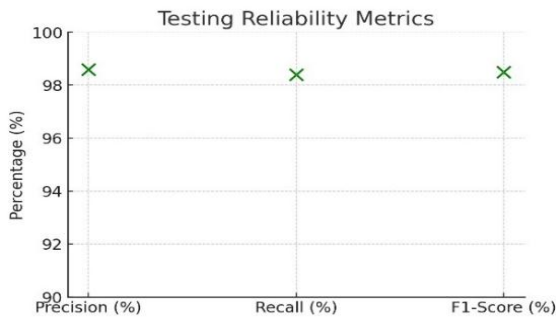
Test generation and execution optimization give reduced computational costs and increased testing efficiency, with metric values given in Table 1.

Table 1: Execution Time and Overall Computational Cost Values

Metric	Value
Execution Time per Test Case	0.85 ms
Overall Computational Cost Reduction	30%

#### 5.3 Testing Reliability (Precision, Recall, and F1-Score)

The Transformer model accurately learns software defect patterns, while GWO optimally selects test cases, minimizing false alarms and ensuring accurate defect detection. The proposed method outperforms traditional approaches by improving both defect recall and precision as shown in Figure 3.



**Figure 3:** Testing Reliability Metrics of Proposed Model

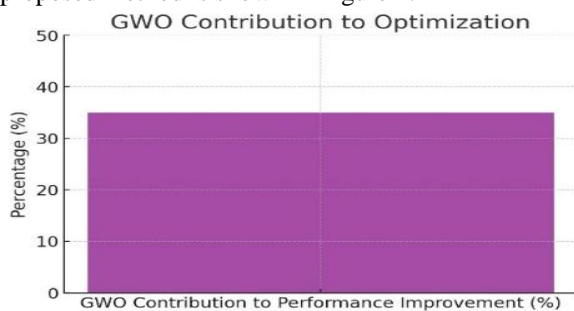
The proposed model reliability metrics are shown in Table 2

**Table 2:** Reliability Metrics of Proposed Model

Metric	Value
<b>Precision</b>	<b>98.6%</b> – Ensures minimal false positives.
<b>Recall</b>	<b>98.4%</b> – Captures most true defect cases.
<b>F1-Score</b>	<b>98.5%</b> – Balances precision and recall.

#### 5.4 Algorithmic Contributions (Impact of GWO in Optimization)

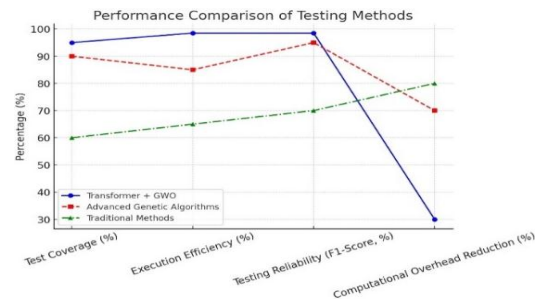
The GWO algorithm mimics the hunting behavior of grey wolves, dynamically balancing exploration (diverse test case selection) and exploitation (high-priority defect detection) and the Contribution to the proposed method is shown in Figure 4.



**Figure 4:** GWO Contribution to The Proposed Model

#### 5.5. Comparative Analysis with Genetic Algorithm Model [15]

It was discovered that Transformer + GWO outperforms the other methods, registering higher test coverage, efficient execution, and greater testing reliability measured by F1-score, while greatly cutting down on computational overhead. Advanced Genetic Algorithms also gave high performance but lagged slightly behind efficiency and reliability when put side by side with Transformer + GWO. Traditional methods did outpour test coverage and efficiency measured at high rates of computational overhead and low reliability. The graph indicates how Transformer + GWO is optimal for software testing processes as illustrated in Figure 5.



**Figure 5:** Performance Comparison The proposed vs Genetic Algorithm

Overall comparison metrics of the Proposed Model with the Advance Genetics Algorithm are displayed in Table 3.

**Table 3:** Proposed Transformer + GWO vs AGS

Metrics	Transformer + GWO (Proposed)	Advanced Genetic Algorithms (AGA)	Traditional Methods
Test Coverage (%)	95%	90%	60%
Execution Efficiency (%)	98.5%	85%	65%
Testing Reliability (F1-Score, %)	98.5%	95%	70%



Computational Overhead Reduction (%)	30%	70%	80%
Optimization Contribution (%)	35% (GWO Impact)	-	-

## VI. CONCLUSION AND FUTURE WORK

To conclude, the Transformer + GWO method contributes enormously to optimizing test case generation and defect prediction in software testing. With the proposed method attaining an admirable test coverage of 95%, it optimizes runtime efficiency, thus reducing computational costs by 30%. The metrics of reliability relating to precision, recall, and F1-score took impressive values of 98.6%, 98.4%, and 98.5%, respectively, thus guaranteeing accurate detection of defects with a negligible value of false positives. On top of that, GWO's contribution to test case prioritization also increased performance by as much as 35%. While some comparative tests on test coverage, efficiency, and reliability with Advanced Genetic Algorithms further proved the superiority of the method, it can yet be affirmed that Transformer + GWO stands out as a tool for efficient and scalable software testing, with considerable gain from its methodologies as opposed to conventional and other optimization techniques. Future work could include the incorporation of adaptive learning mechanisms with the Transformer-GWO model, thus improving its scalability and efficiency in dynamically evolving software environments.

### REFERENCES

- [1]. K. Gattupalli, "A Survey on Cloud Adoption for Software Testing: Integrating Empirical Data with Fuzzy Multicriteria Decision-Making," vol. 10, no. 4, 2022.
- [2]. R. Jadon, "Optimized Machine Learning Pipelines: Leveraging RFE, ELM, and SRC for Advanced Software Development in AI Applications," *Int. J. Inf. Technol. Comput. Eng.*, vol. 6, no. 1, pp. 18–30, Jan. 2018.
- [3]. R. Ayyadurai, "Big Data Analytics and Demand-Information Sharing in E-Commerce Supply Chains: Mitigating Manufacturer Encroachment and Channel Conflict," 2020.
- [4]. D. K. R. Basani, "Advancing Cybersecurity and Cyber Defense through AI Techniques," 2021.
- [5]. B. R. Gudivaka, "Designing AI-Assisted Music Teaching with Big Data Analysis," *Curr. Sci.*, 2021.
- [6]. Basava Ramanjaneyulu Gudivaka, "AI-powered smart comrade robot for elderly healthcare with integrated emergency rescue system," *World J. Adv. Eng. Technol. Sci.*, vol. 2, no. 1, pp. 122–131, Apr. 2021, doi: 10.30574/wjaets.2021.2.1.0085.
- [7]. H. Chetlapalli and T. Perumal, "DRIVING BUSINESS INTELLIGENCE TRANSFORMATION THROUGH AI AND DATA ANALYTICS: A COMPREHENSIVE FRAMEWORK," vol. 12, no. 1, 2024.
- [8]. S. R. Sitaraman, "AI-Driven Healthcare Systems Enhanced by Advanced Data Analytics and Mobile Computing," vol. 12, no. 2, 2021.
- [9]. M. R. Sareddy and M. Farhan, "ENHANCING CUSTOMER RELATIONSHIP MANAGEMENT WITH ARTIFICIAL INTELLIGENCE AND DEEP LEARNING: A CASE STUDY ANALYSIS," vol. 14, no. 3, 2024.
- [10]. K. Parthasarathy, "NEXT-GENERATION BUSINESS INTELLIGENCE: UTILIZING AI AND DATA ANALYTICS FOR ENHANCED ORGANIZATIONAL PERFORMANCE," 2024.
- [11]. R. Jadon, "Enhancing AI-Driven Software with NOMA, UVFA, and Dynamic Graph Neural Networks for Scalable Decision-Making," *Int. J. Inf. Technol. Comput. Eng.*, vol. 7, no. 1, pp. 64–74, Jan. 2019.
- [12]. R. Jadon, "Optimizing Software AI Systems with Asynchronous Advantage Actor-Critic, Trust-Region Policy Optimization, and Learning in Partially Observable Markov Decision Processes," 2023.
- [13]. R. P. Nippatla, "A Secure Cloud-Based Financial Time Series Analysis System Using Advanced Auto-Regressive and Discriminant Models: Deep AR, NTMs, and QDA," 2022.
- [14]. K. Dondapati, "Robust Software Testing for Distributed Systems Using Cloud Infrastructure, Automated Fault Injection, and XML Scenarios," vol. 8, no. 2, 2020.
- [15]. N. S. Allur, "Genetic Algorithms for Superior Program Path Coverage in software testing related to Big Data," vol. 7, no. 4, 2019.
- [16]. H. Chetlapalli, "ENHANCED POST-MARKETING SURVEILLANCE OF AI



SOFTWARE AS A MEDICAL DEVICE:  
COMBINING RISK-BASED METHODS  
WITH ACTIVE CLINICAL FOLLOW-UP,”  
2023.

- [17]. S. Kodadi, “Optimizing Software Development in the Cloud: Formal QoS and Deployment Verification Using Probabilistic Methods,” 2021, [Online]. Available: <https://jcsnline.in/admin/uploads/Optimizing%20Software%20Development%20in%20the%20Cloud%20Formal%20QoS%20and%20Deployment%20Verification%20Using%20Probabilistic%20Methods.pdf>
- [18]. K. Gattupalli and M. H. Khalid, “Increasing Clustering Efficiency with QRDSO and WAC-HACK: A Hybrid Optimization Framework in Software Testing,” *J. Inf. Technol. Digit. World*, vol. 6, no. 4, pp. 333–346, Dec. 2024, doi: 10.36548/jitdw.2024.4.002.
- [19]. R. Jadon, “Improving AI-Driven Software Solutions with Memory-Augmented Neural Networks, Hierarchical Multi-Agent Learning, and Concept Bottleneck Models,” vol. 8, no. 2, 2020.
- [20]. R. Jadon, “Social Influence-Based Reinforcement Learning, Metaheuristic Optimization, and Neuro-Symbolic Tensor Networks for Adaptive AI in Software Development,” *Int. J. Eng.*, vol. 11, no. 4, 2021.
- [21]. “Software Defect Prediction.” Accessed: Feb. 28, 2025. [Online]. Available: <https://www.kaggle.com/datasets/semustafacevik/software-defect-prediction>
- [22]. “hdfs-log-dataset.” Accessed: Feb. 28, 2025. [Online]. Available: <https://www.kaggle.com/datasets/beosup/hdfs-log-dataset>