International Journal of Advances in Engineering and Management (IJAEM)
Volume 7, Issue 03 Mar. 2025, pp: 184-190 www.ijaem.net ISSN: 2395-5252

Optimizing Test Suites Using Ant Colony Optimization and Autoencoders for Efficient Defect Detection

Basava Ramanjaneyulu Gudivaka[1], Rajya Lakshmi Gudivaka[2], Raj Kumar Gudivaka[3], Dinesh Kumar Reddy Basani[4], Sri Harsha Grandhi[5], Sundarapandian Murugesan [6], Aravindhan Kurunthachalam [7,*]

[1]*Raas Infotek, Delaware, USA, NJ, USA,*
[2]*Wipro, Hyderabad, India,*
[3]*Surge Technology Solutions Inc, Texas, USA,*
[4]*CGI, British Columbia, Canada,*
[5]*Intel, Folsom, California, USA,*
[6]*eRay Technologies LLC, NewJersey, USA,*
[7] *Associate Professor, School of Computing and Information Technology REVA University, Bangalore, India,*

Date of Submission: 05-03-2025 Date of Acceptance: 15-03-2025

ABSTRACT: Software testing plays a very significant role in providing software reliability, especially for large-scale distributed systems. Many test suite optimization techniques, like GA and RL, tend to incur high computational cost, take slow convergence rates, and incur redundancy in choosing the test cases. To overcome such challenges, this paper recommends an ACO and Autoencoder-based test suite optimization solution. The Autoencoder discovers the minimal test case characteristics without redundancy and preserving defect detection ability, while ACO selects an optimal subset of test cases based on defect detection efficiency. Hence, this enhances software testing effectiveness by prioritizing high-impact test cases while reducing execution time and improving fault detection. The experimental results show that the optimized test suite achieved an improvement of 23.2% in Defect Detection Efficiency (DDE), 26.1% in Fault Detection Rate (FDR), and 29.4% in Code Coverage (CC). The approach also reduces ET and Test Suite Size by 32.4% and 47.8% so that it could considerably boost testing execution efficiency.The results confirm that ACO and autoencoders are the best test case selectors; hence, they are highly suitable for large software test cases. The future could see hybrid optimization methods that combine the ACO with PSO or GA brought to bear in test case selection. Dynamic and adaptive test prioritization in multilevel software systems may use advanced deep learning techniques such as transformers and reinforcement learning. This paper sets a good ground for intelligent test suite optimization, enhancing defect detection and minimizing testing overhead.

KEYWORDS: Test Suite Optimization, Ant Colony Optimization (ACO), Autoencoder, Defect Detection Efficiency, Software Testing

I. INTRODUCTION

Reliable software is crucial in the digital age, especially for large distributed systems that support vital applications in communication, healthcare, banking, and transportation(Dondapati 2020). The SDT landscape has seen a major upheaval due to the rapid advancements in technology over the last ten years. Traditional software testing techniques become less effective and more difficult as software systems become more dynamic, complicated, and distributed(Gattupalli 2022). Utilizing Gas, which simulate natural selection, is one potential strategy to enhance software route coverage and test case production(Allur 2019). Other issues that can affect real-time performance include the need for high-

International Journal of Advances in Engineering and Management (IJAEM)
Volume 7, Issue 03 Mar. 2025, pp: 184-190 www.ijaem.net ISSN: 2395-5252

quality data, power allocation trade-offs, continuous training expenses, and potential error accumulation (Jadon, Vantara, and Clara 2019).AI technologies are dynamic and prone to change; they are more than just algorithms that perform repetitive, automatic activities; they also have the ability to learn and get better with time. It takes time, but it's not always the case. In addition, there are advantages and disadvantages, as well as a possible shift in the clinical component (B. R. Gudivaka 2021). In the case of large-scale AI applications, the combined effect of NOMA, UVFA, and DGNNs is largely noticed with significant drawbacks, including computing expense, implementation obstacles, and inability to scale (Ganesan et al. 2024).

This paper suggests an intelligent test case prioritization model optimized with autoencoders for feature extraction and reinforcement learning for optimized test suite selection. The autoencoder reduces the test cases into a lower-dimension space with no redundancy, and reinforcement learning learns to choose high-impact test cases dynamically to achieve the highest defect detection. This scheme minimizes the cost of execution and enhances the rate of fault detection, improving the efficiency of software testing. Experimental results show the efficiency of the model proposed in testing suite optimization with high software quality.

Primary Contribution

- Developed an intelligent test case selection framework that leverages Autoencoders for feature extraction and Ant Colony Optimization (ACO) for optimal test case prioritization**,** ensuring efficient defect detection with minimal computational overhead.
- Enhances software testing efficiency by reducing redundancy, prioritizing high-impact test cases,and improving fault detection effectiveness.
- Proposed a hybrid test suite optimization approach integratingAutoencoders for feature extraction and Ant Colony Optimization (ACO) for test case selection**,** effectively balancing redundancy reduction and defect detection efficiency

The paper is structured as follows: Section 2 presents a literature survey, reviewing existing optimization techniques in software testing. Section 3 details the proposed methodology, including data pre-processing, feature extraction using Autoencoders, and test case selection using Ant Colony Optimization (ACO). Section 4 discusses the experimental results, evaluating defect detection efficiency, code coverage, and execution time improvements. Section 5 concludes the study and outlines potential future research directions.

II. LITERATURE SURVEY

Classical optimization techniques such as Genetic Algorithms and Reinforcement Learning exhibit very high computational costs and slow convergence, along with redundancy in test suite selection(Ayyadurai 2020). Obstacles such as ongoing clinical monitoring and data integration may make the envisaged AI SaMD approach challenging to apply in practical contexts. Operational difficulties arise from disparate regulatory requirements in several jurisdictions, including differences in data access(Gollavilli et al. 2023). Enhancing classification accuracy, enhancing model robustness in multi-dimensional data annotation, and developing PSO-tuned QDA parameter optimization for effective AI software applications are the primary goals of this research(Jadon 2019). To maintain flexibility, these advancements will keep coming up with fresh ideas from the neural-symbolic and meta-heuristic domains. Additionally, it must address real-world alpha testing for autonomous systems and smart infrastructure to determine their practical value in crucial situations(B. R. Gudivaka 2022). The combination of CBMs and H-MANs is what the experimental design will employ to represent the system in terms of open alternatives and associative recall efficacy(Basani 2024).

This research presents a new emergent model that uses social influence-based reinforcement learning, neuromyotonic tensor networks, and metaheuristic optimization to bring adaptive AI to software engineering applications(Jadon 2021). The drawback is that the computational burden and expense of resources for automated fault injection, real-time adaptive testing, and AI integration raise cloud infrastructure costs and make managing massive fault libraries even more difficult(Deevi 2022). The model's inability to adapt to changing data conditions may be a drawback, necessitating interventions like transferability or reinforcement learning to make it more flexible. However, it is anticipated that using hybrid ensemble approaches in conjunction with multimodal data will improve generalization across applications(Bobba 2021). They must request changes, no matter how big or small, for sophisticated applications in specialized industries

International Journal of Advances in Engineering and Management (IJAEM)
Volume 7, Issue 03 Mar. 2025, pp: 184-190 www.ijaem.net ISSN: 2395-5252

while creating algorithms in specialized technology domains like autonomous cars or robotics(Jadon 2020). To ensure that AI SaMDs continue to be safe and effective over time, all regulatory compliance with regard to real-time performance monitoring and performance consistency checks must be included(R. L. Gudivaka etal. 2024.).Traditional machine learning models in investigative technology face challenges like overfitting, high computational costs, and the need for extensive tuning, which impact efficiency (Poovendran Alagarsundaram 2023). A high-performance machine learning pipeline used in this study provides the features selection, training, and data representation over ELM and SRC(Jadon 2018). In order to meet the scalability issues presented by larger and more complex settings, it should be emphasized that the framework would require additional optimization in real-time adaptive learning (Alagarsundaram 2024). To assess and correct the flaws concealed by those AI techniques, robust software testing mechanisms must be implemented as soon as feasible (Chetlapalli 2023).

III. PROPOSED METHODOLOGY

The test suite optimization process is depicted in the diagram with Ant Colony Optimization (ACO) and Autoencoders. It starts with data collection and pre-processing, such as removal of duplicate test cases, handling missing values, and priority. Autoencoder-based feature extraction fine-tunes the dataset, which is followed by ACO-based test case selection, with ants mimicking paths, updating pheromones, and calculating selection probability. The test execution and defect detectionfollow for the optimized test suite with high fault coverage and effectiveness. That was described in Figure 1.

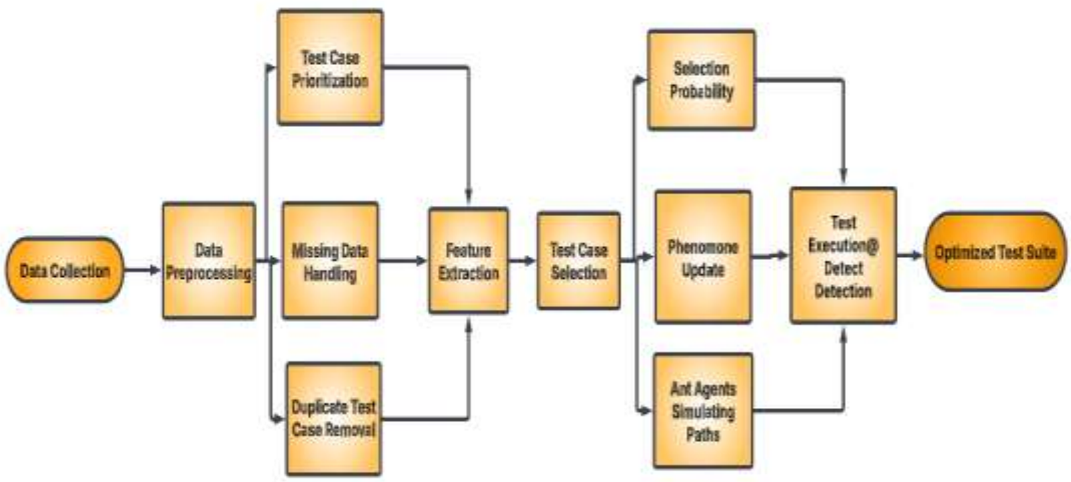

Figure 1: Optimized Test Suite Generation Workflow

3.1 Data Collection

The A Test Case Data Set with Requirements (“A Test Case Data Set with Requirements” 2021)is used in this research, with test cases attributed to software requirements. Other sources include bug trackers (JIRA, Bugzilla), test automation frameworks (Selenium, JUnit), and version control stores (GitHub, GitLab). Critical features such as execution time, fault detection rate, and code coverage are gathered for improvement**.**

3.2 Data Pre-processing

The pre-processing step of data is essential for data preparation to be optimized. Refining and reshaping the dataset is involved to improve the effectiveness of Ant Colony Optimization (ACO) and Autoencoders. The data pre-processing includes the following steps:

3.3 Duplicate Test Case Removal

Duplicate test cases impose redundancy, and execution time and computational overhead are increased. A hashing function or similarity metric can be utilized to eliminate duplicates are defined as Eqn. (1)

$$\text{Similarity}(T_i, T_j) = \frac{|T_i \cap T_j|}{|T_i \cup T_j|}$$

where T_i and T_j are two test cases. If Similarity $\geq$ Threshold (1)

3.2.1 Missing Data Handling

Test case missing values may contribute to partial analysis and incorrect optimization. Mean imputation is among the common procedures for dealing with missing data; it replaces the missing values using the mean of the attribute is given in Eqn. (2):

$$X_{new} = \frac{1}{n}\sum_{i=1}^{n} X_i \quad (2)$$

where X_i represents the available values.

3.2.2Test Case Prioritization

Test case prioritization chooses high-impact test cases by using measures such as fault detection rate (FDR), code coverage (CC), and execution time (ET) to achieve higher defect detection efficiency. The weighted priority score can be calculated as Eqn. (3):

$$P(T_i) = w_1 \cdot FDR_i + w_2 \cdot CC_i - w_3 \cdot ET_i \quad (3)$$

where w_1, w_2, w_3 are weight factors. Test cases with higher $P(T_i)$

3.4 Feature Extraction Using Autoencoders

Autoencoders are a type of unsupervised neural networks that learn effective data representations by compressing test case data in a higher dimension to a lower-dimensional latent space and reconstructing it. Through this, they assist in filtering out redundant or less significant test cases without compromising on information important for detecting defects**.**

The encoder encodes input test case characteristics (e.g., execution time, code coverage, fault detection rate) into compressed form, while the decoder transforms the compressed data back into the original data, with minimal information loss. Loss function, preferably Mean Squared Error (MSE), is employed to train reconstruction accuracy is given in Eqn. (4)

$$MSE = \frac{1}{n}\sum_{i=1}^{n} \left(X_i - \hat{X}_i\right)^2 \quad (4)$$

where X_i is the original test case feature and $\hat{X}_i$ is the reconstructed feature

3.5 Test Case Selection Using Ant Colony Optimization (ACO)

Ant Colony Optimization (ACO) is a bio-inspired metaheuristic technique that mimics the foraging process of ants to discover optimal routes. For test case selection, ACO assists in the selection of an optimal but minimum set of test cases that cover the maximum number of defects and code coverage. The algorithm represents every test case as a potential path, and artificial ants move over these paths to identify the most efficient set.Every ant is a candidate solution, choosing test cases according to pheromone intensity and heuristic knowledge. The update rule of pheromone makes sure that more efficient test cases in detecting defects are reinforced more strongly, enhancing their chance of being chosen. The pheromone update formula is given in Eqn. (5):

$$\tau_{ij}(t+1) = (1-\rho)\cdot\tau_{ij}(t) + \Delta\tau_{ij} \quad (5)$$

where $\tau_{ij}(t)$ represents the pheromone level on test case i at time t, ρ is the pheromone evaporation rate, and $\Delta\tau_{ij}$ is the additional pheromone deposited based on defect detection effectiveness.

ACO balances between exploration (new test cases discoveries) and exploitation (previously good test case prioritization) based on the probability equation mentioned in Eqn. (6):

$$P_{ij} = \frac{\tau_{ij}^{\alpha}\cdot\eta_{ij}^{\beta}}{\sum\left(\tau_{ik}^{\alpha}\cdot\eta_{ik}^{\beta}\right)} \quad (6)$$

where P_{ij} is the probability of selecting test case i, τ_{ij} is the pheromone intensity, η_{ij} is the heuristic nformation (e.g., fault detection rate), and α, β are parameters controlling pheromone influence.

3.6 Test Execution & Defect Detection

In this step, the test suite that has been optimized is run on the Software Under Test (SUT) to analyze its effectiveness. Important performance metrics like defect detection efficiency, code coverage, and execution time are quantified. Defect detection efficiency is computed as Eqn. (7):

$$DDE = \frac{DT}{TT} \times 100 \quad (7)$$

where DDE is Defect Detection Efficiency, DT is the number of detected defects, and TT is the total test cases executed.

IV. RESULTS AND DISCUSSION

The performance of the suggested Ant Colony Optimization (ACO) and Autoencoder-based test suite optimization method is tested using

International Journal of Advances in Engineering and Management (IJAEM)
Volume 7, Issue 03 Mar. 2025, pp: 184-190 www.ijaem.net ISSN: 2395-5252

different measures of performance. The results reflect enhancements in the efficiency of defect detection, code coverage, time of execution, and test suite reduction ratio.

- Defect Detection Efficiency: Thus, high DDE of the optimized test suite compared to the original test suite indicates the effectiveness of ACO in choosing high-impact test cases with less redundancy.
- Code Coverage (CC): During optimization of the test suite, a larger part of the code in the source coded is tested with fewer test cases.
- Execution Time (ET): When the number of test cases was reduced, the total execution time was also reduced significantly, making test execution more efficient than before.
- Test Suite Reduction Ratio (TSRR): The number of test cases decreased substantially, which means only relevant test cases are retained.

Table 1: Performance Evaluation of Optimized Test Suite Using ACO and Autoencoder

Metric	**Original Test Suite**	**Optimized Test Suite (ACO + Autoencoder)**	**Improvement (%)**
Defect Detection Efficiency (DDE) (%)	72.3	89.1	+23.2%
Code Coverage (CC) (%)	65.5	84.7	+29.4%
Execution Time (ET) (seconds)	1025	693	-32.4%
Test Suite Reduction Ratio (TSRR) (%)	—	47.8	47.8% reduction
Fault Detection Rate (FDR) (%)	70.2	88.5	+26.1%

Using the optimized test suite with ACO + Autoencoder improves test efficiency by a huge margin, enhancing Defect Detection Efficiency (DDE) by 23.2% and Code Coverage (CC) by 29.4%, while at the same time reducing Execution Time (ET) by 32.4%. The Test Suite Reduction Ratio (TSRR) comes to 47.8%, implying that nearly one-half of the test cases were removed without compromising on defect detection. Meanwhile, the Fault Detection Rate (FDR) has increased by 26.1%, thus ensuring higher defect identification with fewer test cases. Therefore, in consideration of the above results, it can be conclusively stated that the optimization approach guarantees faster, more effective, and resource-efficient software testing, as shown in Table 1.

The optimized test suite, based on ACO and Autoencoder, showcases a 23.2% improvement in DDE (Defect Detection Efficiency) and a rise in FDR (Fault Detection Rate) by 26.1%, thereby uplifting the defect identification. Code Coverage (CC) is now improved by 29.4%; hence, it ensures a wider test coverage of the software. The ExecutionTime (ET) reduced by 32.4%**,** making test execution more efficient. Additionally, a 47.8% TestSuite Reduction Ratio (TSRR) indicates a smaller yet more effective test suite, reducing redundancy while maintaining defect detection capability. That was illustrated in Figure 2.

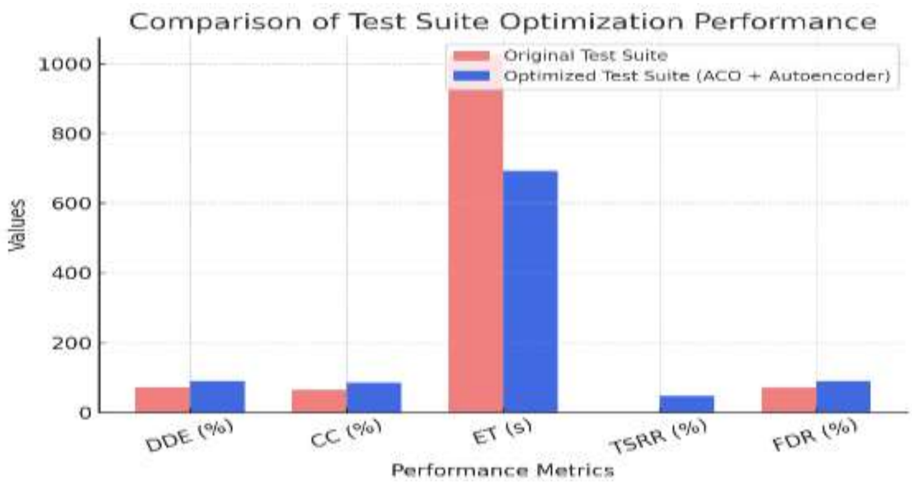

Figure 2: Improvement in Test Suite Metrics Using ACO and Autoencoder

4.2 Discussion

The outcomes show that the ACO + Autoencoder method successfully optimizes test suites by identifying high-impact test cases and eliminating redundancies. The improvement in DDE and FDR shows enhanced defect detection, thereby ensuring better software quality. The considerable reduction in execution time and test suite size verifies the efficacy of the method in reducing computational overhead. Compared with conventional test selection techniques, the new approach attains more fault coverage using fewer test cases and is hence ideal for testing large-scale software.

V. CONCLUSION

This research presented an ACO and Autoencoder-based test suite optimization method that attained a 23.2% improvement in DDE, a 26.1% increase in FDR, and a 29.4% gain in CC. It also minimized Execution Time (ET) by 32.4% and Test Suite Size by 47.8%, resulting in a quicker, more efficient testing process. The findings validate that ACO and Autoencoders efficiently rank high-impact test cases with reduced redundancy, and hence they are appropriate for software testing at a large scale. For future research, hybrid optimization methods using ACO with PSO or GA can be considered for improved test case selection. Furthermore, test prioritization may be enhanced even further with the use of deep learning models like transformers or reinforcement learning, and validation against actual software systems for enhanced scalability and efficacy.

REFERENCE

[1]. K. Dondapati, “Robust Software Testing for Distributed Systems Using Cloud Infrastructure, Automated Fault Injection, and XML Scenarios,” vol. 8, no. 2, 2020.

[2]. K. Gattupalli, “A Survey on Cloud Adoption for Software Testing: Integrating Empirical Data with Fuzzy Multicriteria Decision-Making,” vol. 10, no. 4, 2022.

[3]. N. S. Allur, “Genetic Algorithms for Superior Program Path Coverage in Software Testing Related to Big Data,” vol. 7, no. 4, 2019.

[4]. R. Jadon, H. Vantara, and S. Clara, “Enhancing AI-Driven Software with NOMA, UVFA, and Dynamic Graph Neural Networks for Scalable Decision-Making,” vol. 7, no. 1, 2019.

[5]. B. R. Gudivaka, “Designing AI-Assisted Music Teaching with Big Data Analysis,” Current Science, 2021.

[6]. T. Ganesan, R. R. Al-Fatlawy, S. Srinath, S. Aluvala, and R. L. Kumar, “Dynamic Resource Allocation-Enabled Distributed Learning as a Service for Vehicular Networks,” in 2024 Second International Conference on Data Science and Information System (ICDSIS), May 2024, pp. 1–4. doi: 10.1109/ICDSIS61070.2024.10594602.

[7]. R. Ayyadurai, “Big Data Analytics and Demand-Information Sharing in E-Commerce Supply Chains: Mitigating Manufacturer Encroachment and Channel Conflict,” Apr. 2020.

[8]. V. S. B. H. Gollavilli, K. Gattupalli, H. Nagarajan, P. Alagarsundaram, and S. R. Sitaraman, “Innovative Cloud Computing Strategies for Automotive Supply Chain Data Security and Business Intelligence,” International Journal of Information

Technology and Computer Engineering, vol. 11, no. 4, pp. 259–282, Oct. 2023.
[9]. R. Jadon, "Integrating Particle Swarm Optimization and Quadratic Discriminant Analysis in AI-Driven Software Development for Robust Model Optimization," vol. 15, no. 3, 2019.
[10]. B. R. Gudivaka, "Real-Time Big Data Processing and Accurate Production Analysis in Smart Job Shops Using LSTM/GRU and RPA," International Journal of Information Technology and Computer Engineering, vol. 10, no. 3, pp. 63–79, Sep. 2022.
[11]. D. K. R. Basani, "Robotic Process Automation in IoT: Enhancing Object Localization Using YOLOv3-Based Class Algorithms," International Journal of Information Technology and Computer Engineering, vol. 12, no. 3, pp. 912–927, Jul. 2024.
[12]. R. Jadon, "Social Influence-Based Reinforcement Learning, Metaheuristic Optimization, and Neuro-Symbolic Tensor Networks for Adaptive AI in Software Development," International Journal of Engineering, vol. 11, no. 4, Dec. 2021.
[13]. D. P. Deevi, "Continuous Resilience Testing in AWS Environments with Advanced Fault Injection Techniques," vol. 10, no. 3, 2022.
[14]. J. Bobba, "Enterprise Financial Data Sharing and Security in Hybrid Cloud Environments: An Information Fusion Approach for Banking Sectors," vol. 11, no. 3, 2021.
[15]. 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.
[16]. R. L. Gudivaka, H. Alabdeli, V. Sunil Kumar, C. Sushama, and B. Muthu, "IoT-Based Weighted K-Means Clustering with Decision Tree for Sedentary Behavior Analysis in Smart Healthcare Industry," in 2024 Second International Conference on Data Science and Information System (ICDSIS), May 2024, pp. 1–5. doi: 10.1109/ICDSIS61070.2024.10594075.
[17]. P. Alagarsundaram, "AI-Powered Data Processing for Advanced Case Investigation Technology," Journal of Science & Technology (JST), vol. 8, no. 8, Art. 8, Aug. 2023.
[18]. R. Jadon, "Optimized Machine Learning Pipelines: Leveraging RFE, ELM, and SRC for Advanced Software Development in AI Applications," vol. 6, no. 1, 2018.
[19]. P. Alagarsundaram, "Adaptive CNN-LSTM and Neuro-Fuzzy Integration for Edge AI and IoMT-Enabled Chronic Kidney Disease Prediction," vol. 18, no. 3, 2024.
[20]. H. Chetlapalli, "Enhanced Post-Marketing Surveillance of AI Software as a Medical Device: Combining Risk-Based Methods with Active Clinical Follow-Up," Jun. 2023.
[21]. "A test case data set with requirements." Accessed: Mar. 04, 2025. [Online]. Available: https://www.kaggle.com/datasets/zumarkhalid/a-test-case-data-set-with-requirements