AUTOMATIC GENERATION OF SOFTWARE ENGINEERING TEST CASES AND CONSTRUCTION OF COVERAGE IMPROVEMENT MODEL BASED ON GENETIC ALGORITHM OPTIMIZING NEURAL NETWORK

Wan Ming
School of Modern Information Technology, Xuchang Vocational and Technical College, Xuchang 461000, China

Abstract - The scale of software system continues to expand, and the limitations of manual writing of test cases are gradually exposed in terms of coverage integrity and resource consumption. Focusing on the requirements of automatic test case generation and coverage improvement, this paper constructs a test case generation model based on genetic algorithm optimization neural network, which combines coverage behavior prediction with search optimization process. The model uses vectorization test case coding method, and uses neural network to predict the coverage trend of sentences and branches, thus guiding the evolution of genetic search in high-value input areas. Experiments are carried out on software modules with the scale of 120,000 to 180,000 lines of code, and the test time is limited to 60 minutes. The experimental results show that the effective ratio of test cases generated by the model is stable above 70%, the sentence coverage is maintained in the range of 82% to 83%, the branch coverage is about 70%, and the coverage fluctuation is controlled within 0.4%. The results show that the model has good coverage improvement ability and running stability under the condition of limited test resources, and is suitable for software test scenarios with complex control logic.

Keywords: Genetic algorithm; Neural network; Software engineering testing; Automatic generation of test cases; Coverage improvement.

1. Introduction

In recent twenty years, the scale of software system has continued to expand, and the single application has gradually evolved into a distributed, service-oriented and highly concurrent form. The number of code lines usually reaches millions, and there is a complex call relationship between functional modules. The traditional way of writing test cases by relying on manual experience has been difficult to cover a large number of conditional branches and abnormal paths, the test cycle has been significantly lengthened, and the labor cost has been rising continuously. Taking a single version iteration of business system as an example, the core module often involves dozens of input parameter combinations, and it is difficult to cover it in a limited time only by manual design. Although the existing random testing has a certain degree of automation, the input distribution is not targeted, the generated use cases are unstable in path coverage and defect reach, and a large number of repeated or invalid use cases often appear. Search method can alleviate this problem to some extent, but the search direction depends on manual rules, and it is easy to fall into local optimum when facing complex program structure. The dual pressure of software quality requirements and delivery rhythm has prompted the evolution of testing activities to automatic generation and intelligent optimization, and building a test case generation mechanism with learning ability and adaptive characteristics has become an important background of current software engineering testing research.

Neural network shows good adaptability in nonlinear mapping and feature association modeling, and can learn the implicit relationship between input features and coverage behavior from historical test data, thus providing predictive basis for test decision-making. However, when the neural network is used alone, the training results of the model are obviously affected by the initial weight and sample distribution, and the output stability fluctuates. Genetic algorithm has global search

dx.doi.org/10.17683/ijomam/issue24.10

ability, maintains population diversity in complex solution space, and is suitable for test case combination optimization and parameter optimization, but the search efficiency is easily limited by evaluation function in high-dimensional space. The fusion of the two methods helps to form a complementary structure: neural network undertakes coverage behavior evaluation and trend judgment, and genetic algorithm is responsible for selecting, crossing and mutating in the candidate test case space, which promotes the search direction to evolve towards high coverage areas. This fusion mode can reduce the proportion of blind search and improve the concentration of test cases in sentence coverage and branch coverage. For software engineering testing, this idea not only improves the quality of use case generation, but also introduces data-driven mechanism for testing process, and enhances the adaptability of testing activities in complex system scenarios.

Focusing on the requirements of automatic test case generation and coverage improvement in software engineering, this paper plans to build a test case generation model based on genetic algorithm and neural network optimization. At the method level, a unified test case coding method is established, the input parameters are mapped into a vector structure that can participate in genetic operation, and the coding length and value range are limited to adapt to common business system interfaces. At the model level, neural network is introduced to model the relationship between test input and coverage behavior to evaluate the potential contribution of candidate use cases in sentence coverage and branch coverage. On the optimization level, fitness evaluation rules including coverage and use case difference are designed to guide the genetic search process to evolve to high-value test areas. At the experimental level, software modules with the scale of 100,000 to 500,000 lines of code are selected to verify the performance of the model in terms of the number of test cases generated, the effectiveness ratio and the coverage improvement range. The coverage evaluation index is limited to the range of sentence coverage and branch coverage. The ultimate goal is to form a set of reusable and extensible intelligent test case generation methods to provide stable support for complex software system testing.

2. Literature Review

2.1 Research on Genetic Algorithm

As a global optimization method based on natural selection and genetic mechanism, genetic algorithm has formed a systematic research system in its theoretical framework and application form. Katoch et al. (2021) pointed out in a systematic review of the development of genetic algorithm that this method takes group search as its basic feature, relies on selection, crossover and mutation operations to maintain the ability to explore solution space, and shows stable performance in continuous optimization and combinatorial optimization problems, and the research gradually shifts from basic operator design to parameter adaptive and hybrid optimization [1]. Genetic algorithm has evolved from a general search tool to an optimization framework that can be customized for specific problem structures.

In the field of solving complex engineering models, Shi (2020) introduced adaptive grey genetic algorithm based on large-scale building structure optimization problem to improve the problem of insufficient convergence speed of traditional genetic algorithm in high-dimensional search. It is considered that introducing dynamic adjustment mechanism can help alleviate the early shock and late stagnation of search [2]. The research shows the research trend of genetic algorithm's flexible adjustment of search strategy in complex constraint environment.

Focusing on the optimization of the algorithm structure, Chang (2020) put forward the idea of partial optimization genetic algorithm in the study of complex network community identification, and simplified the selection and mutation stages. The study pointed out that this method is conducive to reducing invalid search and improving the stability of the solution [3]. Relevant conclusions reflect that researchers gradually pay attention to the matching relationship between the internal structure of genetic algorithm and the characteristics of the problem.

In the data-driven application scenario, Shi (2021) introduced the improved genetic algorithm into the artificial intelligence data mining system. The research shows that the genetic algorithm has strong global exploration ability in the task of rule search and feature combination, and is suitable for dealing with nonlinear and multimodal data spaces [4]. This view emphasizes the applicability of genetic algorithm in complex data analysis tasks. To solve the resource scheduling problem, Sun et al. (2020) built an improved genetic algorithm model in the research of cloud computing task scheduling, and the research pointed out that by adjusting the fitness evaluation method, the algorithm showed more stable search behavior in multi-objective scheduling scenarios [5]. This study shows that the application of genetic algorithm in multi-objective optimization problems is gradually mature. In the field of industry data analysis, Wang et al. (2020) take the intelligent analysis of financial industry data as the object, and think that genetic algorithm has strong adaptability in high-dimensional feature selection and model parameter optimization, and is suitable for dealing with complex business data structures [6]. The

related work of genetic algorithm has shifted from the research of basic mechanism to the stage of structural optimization and integration application oriented to specific problems.

2.2 Research on Genetic Algorithm to Optimize Neural Network

The fusion research of genetic algorithm and neural network mainly focuses on parameter optimization, structure adjustment and learning process improvement. Zhou et al. (2020) built a portfolio model combining neural network and genetic algorithm under the background of big data, and the research content focused on the global search of neural network weights and thresholds by using genetic algorithm. The author thinks that this fusion method is helpful to alleviate the problem of sensitivity to initial parameters in the training process of neural network and maintain stable prediction performance under complex data distribution conditions [7].

According to the learning ability of neural network itself, Kotyrba et al. (2022) analyzed the influence of genetic algorithm on the learning process of neural network from the perspective of algorithm mechanism, covering the role of genetic algorithm in weight initialization and training path selection. The author points out that genetic search mechanism can expand the exploration range of neural network parameter space, reduce the probability of falling into local optimum and improve the network convergence behavior [8].

On the specific application level, Zheng (2022) combined the improved genetic algorithm with BP neural network for air quality assessment, and the research content focused on the synchronous optimization of network weights and thresholds by genetic algorithm. The author thinks that this method shows high prediction accuracy and model stability in the modeling of nonlinear environmental indicators [9]. Focusing on the optimization of network structure, Yang (2020) introduced genetic algorithm into the study of indoor space combination to optimize the neural network structure, and the research content involved node configuration and parameter adjustment strategy. The author points out that genetic algorithm has a strong global nature in network structure search, which is conducive to improving the expression ability of the model to complex spatial relations [10].

In the field of engineering materials, Jiang (2025) built an optimization model of mortar mixture ratio based on neural network and genetic algorithm, and the research content focused on the optimization ability of genetic algorithm under multi-parameter coupling conditions. The author thinks that the fusion model can maintain good search stability in high-dimensional parameter space [11]. In order to meet the demand of high-precision prediction, Qiang et al. (2024) combined the improved intelligent optimization algorithm with the multi-layer perceptron neural network for debris flow scale prediction. The research content reflects the adaptability of genetic algorithm in neural network parameter optimization, and the author thinks that the hybrid optimization strategy is helpful to improve the prediction consistency of the model in complex nonlinear scenes [12].

2.3 Research on Software Engineering Testing

The research of software engineering testing has gradually shifted from the traditional function verification to the comprehensive research of data, process and tool system. Kim (2020) focuses on the software engineering problems under the background of data analysis, focusing on the engineering complexity faced by data-intensive systems in the development and testing stages. The author thinks that testing activities need to be closely combined with data processing flow to support the stable operation of the system under the conditions of scale expansion and algorithm update [13]. At the interface level of software engineering teaching and practice, Sasmito et al. (2021) took the software engineering course as the research object, and analyzed the influence of teaching materials on the understanding and application of testing methods. The research content involved the testing process, tool cognition and practical training arrangement. The author pointed out that systematic software engineering teaching was helpful to strengthen the standardized implementation of testing methods in actual development [14].

Around the practice of automated testing, Mischke et al. (2022) compared manual testing with automated testing in the development of RCE, covering the selection of testing strategies and the application of tools. The author thinks that the parallel use of automated testing and manual testing is helpful to improve the integrity of test coverage and process controllability in complex scientific research software [15]. In view of the current situation of scientific research software testing, Eisty and Carver (2022) summarized the characteristics of testing practice through questionnaire and literature analysis, and the research contents involved the selection of testing methods, the frequency of tool use and common difficulties. The author pointed out that there were obvious differences in the automation degree and coverage depth of scientific research software testing, and the testing process was highly heterogeneous [16].

In the field of mobile application software engineering research, Zein et al. (2023) analyzed the test-related research results based on the systematic

review method, covering the distribution of test technology, evaluation indicators and research methods. The author believes that the existing research is still in a decentralized state in terms of empirical depth and tool unity [17]. In the follow-up expansion research, Eisty et al. (2025) further combed the testing methods, tools and practices of scientific research software, and the research content emphasized the adaptation of testing activities in different software types. The author thinks that the selection of testing methods is obviously influenced by software scale, domain characteristics and development mode, and has not yet formed a unified paradigm [18].

2.4 Research on the Improvement of Software Testing Coverage

The research on improving software test coverage focuses on the strategy of test case selection, sequencing and generation, with the goal of expanding the reach of program structure and reducing resource waste. Wang (2024) introduced the method of combining greedy algorithm weight ranking in the regression test scenario, and studied the relationship between the execution order of test cases and coverage contribution. The author thinks that weight ranking based on coverage contribution is helpful to improve the coverage efficiency in the regression test stage within a limited number of executions [19]. In the research of coverage-oriented software evolution, Kaur et al. (2025) combined the software growth model of coverage execution orientation with the release planning driven by genetic algorithm, and the research content involved the dynamic changes of coverage data in the process of version evolution. The author points out that coverage execution information can be used as an important basis for release decision to coordinate reliability evaluation and test resource allocation [20].

In order to solve the problem that defects are hard to reach, Zheng et al. (2024) put forward an automated testing method that integrates Olympic optimization algorithm. The research content focuses on the analysis of the ability to cover deep paths and hidden defects. The author thinks that the hybrid optimization strategy shows stronger coverage expansion ability in the depth of path exploration [21]. In the research of regression test priority, Alrawashdeh et al. (2021) put forward a test case priority ranking method based on improved genetic algorithm, and the research content revolved around the combination of coverage information and genetic operators. The author thinks that introducing coverage index into genetic search process will help to maintain a high coverage growth rate in regression test stage [22].

Around the systematic utilization of coverage spectrum information, Bertolino et al. (2021) put forward an adaptive test case allocation, selection and generation method. The research content emphasizes the joint use of coverage spectrum and running profile. The author thinks that coverage information has a continuous guiding role in the test life cycle and is suitable for dynamically adjusting test strategies [23]. In the direction of automatic test generation, Arasteh et al. (2025) integrated machine learning method and swarm intelligence algorithm to carry out test case generation research, focusing on coverage path expansion and defect discovery ability. The author thinks that combining learning mechanism and swarm search strategy is helpful to improve the coverage integrity of complex program structures [24].

2.5 Literature Review

The existing research focuses on genetic algorithm, neural network and software engineering testing, and gradually forms an intersection between the automatic generation of test cases and the improvement of coverage. On the research of genetic algorithm, most scholars pay attention to the improvement of algorithm operator, parameter adaptation and stability performance in complex search space. The research conclusion shows that genetic algorithm has strong global search ability in high-dimensional and multimodal problems, but search efficiency and result stability depend on fitness function design and problem modeling method. Focusing on the research of optimizing neural network by genetic algorithm, the related work mainly focuses on the optimization of weight, threshold and network structure. The research results show that genetic algorithm is helpful to alleviate the local convergence and initial sensitivity problems in the training process of neural network, but the application scenarios are mostly focused on prediction and evaluation tasks, and the systematic modeling in the field of software testing is still limited. In the research of software engineering testing, the existing literatures are launched from the perspectives of automatic testing, regression testing and scientific research software testing practice, revealing the problems of strong heterogeneity of testing process and scattered tools and methods. Coverage is widely used as a measure of test adequacy, but its mechanism to guide the generation of test cases is not unified. Aiming at the research of coverage improvement, most of the existing achievements start with test case sequencing, priority allocation and hybrid intelligent optimization methods, emphasizing the utilization value of coverage information under the condition of limited test resources, but the collaborative relationship between coverage optimization process

and learning model is still lacking in-depth discussion.

3. Research Methods

3.1 Genetic Algorithm to Optimize the Overall Design of Neural Network test Case Generation Model

The test case generation model constructed in this study consists of test case presentation layer, neural network prediction layer, genetic optimization layer and coverage feedback layer, as shown in Figure 1, and the overall structure revolves around the closed-loop logic of "prediction-screening-optimization-feedback". When the model is running, the initial test cases enter the neural network prediction layer in vector form, and the neural network outputs the corresponding coverage behavior prediction results according to the existing test history and program structure characteristics. The prediction result is not directly used as the final test conclusion, but as an important basis for individual evaluation in genetic algorithm. The genetic algorithm layer selects, crosses and mutates the test cases according to the predicted coverage information and the actual execution coverage data, and promotes the evolution of the test case set to the high coverage direction in multiple iterations. The coverage feedback layer records the real coverage after each test, and sends this information back to the prediction layer to update the neural network parameters. The overall operation of the model emphasizes the continuous flow of information between different modules to avoid the test case generation process from the actual behavior of the program.

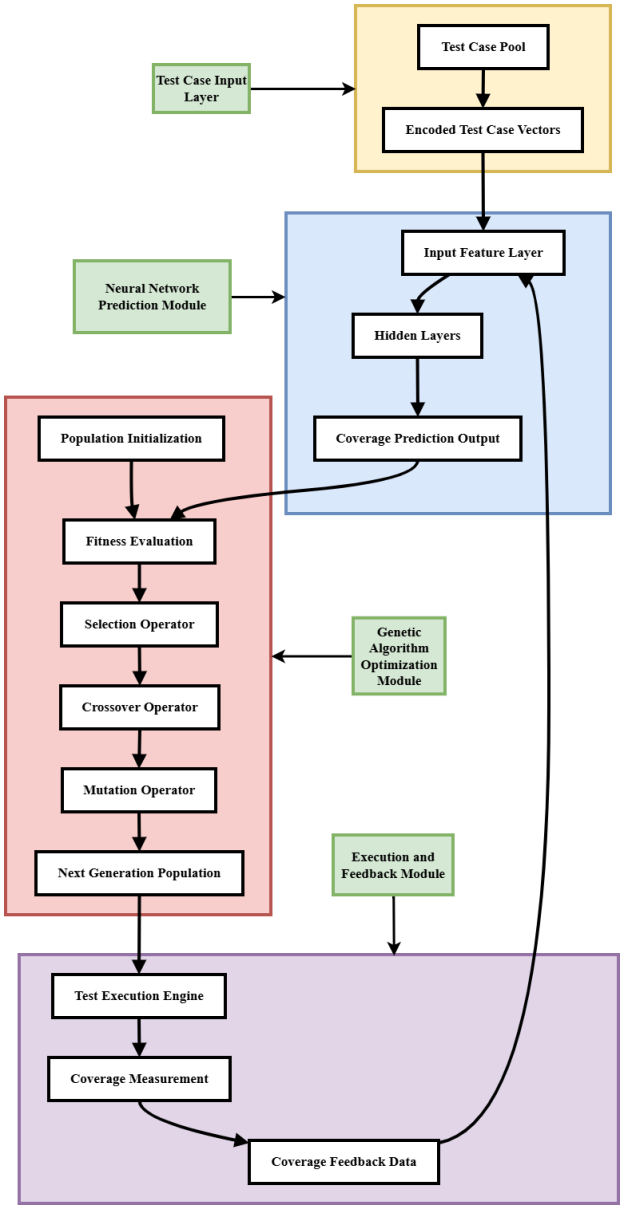

Figure 1: Overall architecture of ga-optimized neural network for test case generation

3.2 Test Case Coding and Search Space Construction

Test cases are represented in the model in the form of fixed-length vectors, and each vector corresponds to a complete test input. Each dimension in the vector corresponds to the program input parameters one by one. Discrete parameters are coded by integers, and continuous parameters are normalized according to the actual value interval. The search space consists of the combination of all parameters, and the boundary range is determined according to the interface document of the tested program.

For example, integer parameters are limited in the [0,100] range, floating-point parameters are limited in the [0.0,1.0] range, and string parameters are converted into enumeration index values. The coding method keeps the structure stable and avoids illegal input in genetic operation.

Formula (1) Test case coding vector representation model:

$$X = (x_1, x_2, \ldots, x_n) \tag{1}$$

x_i represents the coded value of the i the input parameter, n represents the total number of program input parameters, and X represents the position of a single test case in the search space.

Table 1 is the sample data of test case coding structure.

Table 1. Sample Data of Test Case Coding Structure

Use case number	Parameter 1	Parameter 2	Parameter 3	Parameter 4
TC01	twelve	0.45	three	one
TC02	87	0.10	five	0

3.3 Neural Network Test Coverage Behavior Prediction Model Design

The coverage behavior prediction model adopts multi-layer feedforward neural network structure, and the number of nodes in the input layer is consistent with the coding vector dimension of the test case, and the number of nodes in the output layer corresponds to the number of coverage indicators. Network training samples are derived from historical test execution data, and each sample contains test case codes and corresponding actual coverage results. In the process of network training, the error value is used to measure the deviation between the prediction result and the real coverage, as a basis for parameter updating. Formula (2) Neural network output mapping model:

$$\hat{y} = f\left(\sum_{i=1}^{n} w_i x_i + b\right) \tag{2}$$

Where x_i represents the i the coded component of the test case, w_i represents the corresponding weight parameter, b represents the offset term, $f(\cdot)$ represents the activation function, and $\hat{y}$ represents the predicted coverage value. Formula (3) Prediction error calculation model:

$$E = \frac{1}{m}\sum_{j=1}^{m}(y_j - \hat{y}_j)^2 \tag{3}$$

y_j represents the actual coverage of the j the test case, $\hat{y}$ represents the predicted coverage, and m represents the number of samples. Table 2 is the neural network prediction results and actual coverage results.

Table 2. Neural Network Prediction Results and Actual Coverage Results

Use case number	Forecast coverage rate	Actual coverage rate	error
TC01	0.72	0.75	0.03
TC02	0.61	0.58	0.03

3.4 Genetic Algorithm Optimization Strategy and Fitness Function Design

Genetic algorithm takes the test case coding vector as the individual representation, the population size is set to 50, the crossover probability is set to 0.8, and the mutation probability is set to 0.05. The selection strategy adopts roulette mechanism based on fitness sorting, the crossover operation adopts single-point crossover, and the mutation operation is limited in the legal range of parameters. The fitness function comprehensively considers the difference between prediction coverage and test cases to avoid the rapid convergence of the population to the local area. Formula (4) Test Case Comprehensive Fitness Function:

$$F = \alpha \cdot \hat{C} + \beta \cdot D \quad (4)$$

$\hat{C}$ represents the predicted coverage of the neural network, D represents the average difference between the test case and other use cases in the population, α and β represent the coverage weight and the difference weight respectively, and the value ranges are [0.6,0.8] and [0.2,0.4] respectively.

Table 3 is the parameter setting and description of genetic algorithm.

Table 3. Parameter Setting and Description of Genetic Algorithm

Parameter name	numerical value	explain
Population size	50	Number of test cases per generation
Cross probability	0.8	Control recombination frequency
Mutation probability	0.05	Control diversity

3.5 Model Collaborative Operation Mechanism and Optimization Process

During the operation of the model, all modules work together in a fixed order. As shown in Figure 2, after the test cases enter the prediction module, the prediction results are passed to the genetic algorithm module to participate in individual evaluation, and the new test case set generated by the genetic algorithm enters the program execution stage, and the real coverage results are fed back to the prediction module to update parameters.

This process is repeated after each iteration, until the growth rate of coverage is less than 0.5% or the number of iterations reaches 30 rounds.

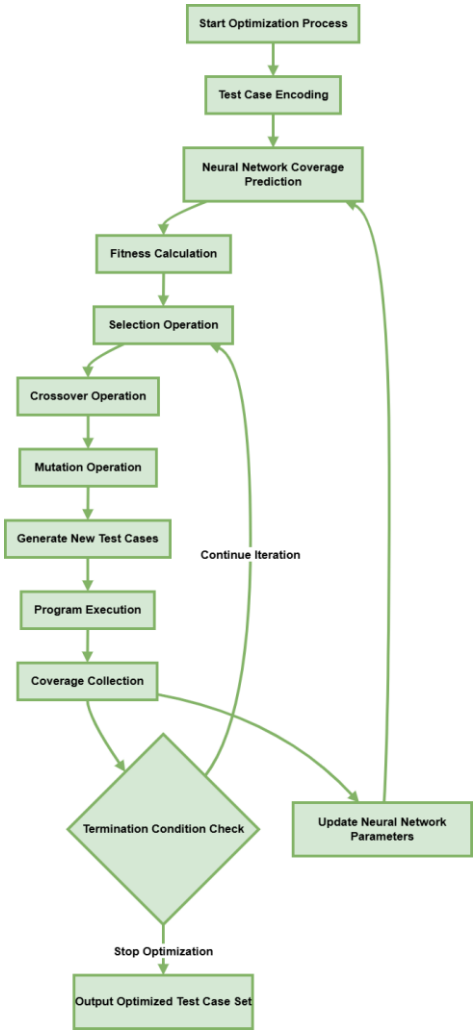

Figure 2: Collaborative Optimization Workflow of Test Case Generation

3.6 Evaluation Index and Result Expression Method of Coverage Improvement

Coverage evaluation uses sentence coverage as the core index, and counts the coverage changes before and after the test. The coverage improvement rate is used to quantify the optimization effect of the model, which is uniformly expressed in the interval of [0,100%]. Formula (5) Calculation model of test coverage improvement rate:

$$\Delta C = \frac{C_{opt} - C_{base}}{C_{base}} \times 100\% \tag{5}$$

C_{base} represents the initial test case set coverage, C_{opt} represents the optimized test case set coverage, and ΔC represents the percentage increase of coverage. This evaluation method is used to compare the results of different methods in the experimental stage, and the statistical granularity of coverage is limited to sentence level to ensure the consistency and comparability of the results.

4. Experimental Evaluation

4.1 Experimental Object, Environment Configuration and Evaluation Index Setting

The experimental object selects the software module with clear input parameter structure and complex control logic to ensure that the automatic generation of test cases and coverage analysis are operable and representative. The tested program comes from the core computing and process control module in the open source business system, and the code size is controlled between 120,000 and 180,000 lines, including conditional branching, loop structure and exception handling logic, which is suitable for statement-level and branch-level coverage analysis. The program input interface contains 8 to 12 parameters, including integer, floating point and enumeration types. The range of integer parameters is limited to 0 to 100, the range of floating point parameters is limited to 0.0 to 1.0, and the number of enumeration parameters is 3 to 6. There is no external dependent service in the test case execution environment to avoid the interference of uncertain factors in the running results.

The configuration of experimental environment revolves around three links: test case generation, model training and program execution. The hardware environment adopts Intel Core i7 processor, clocked at 3.4 GHz, with 16 GB of memory capacity and 512 GB of solid-state storage space.

The software environment is based on Ubuntu 20.04 operating system, the Java running environment version is JDK 11, the test framework is JUnit 5.9, and the coverage statistics tool is JaCoCo 0.8.10. The neural network model runs in Python 3.9 environment. TensorFlow 2.10 is selected as the deep learning framework, the training batch size is set to 64, the learning rate is set to 0.001, and the maximum training rounds are set to 50. The operating parameters of genetic algorithm are consistent with the model design chapter, the population size is fixed at 50, the maximum number of iterations is set at 30, and the total running time of a single experiment is controlled within 60 minutes to ensure that different methods are compared under the same resource conditions.

Evaluation indicators are set around test coverage effect, test case quality and generation efficiency. Coverage indicators include statement coverage and branch coverage, and the statistical granularity of coverage is consistent, both of which are calculated based on the execution result of a single complete test. The quality of test cases is measured by the ratio of effective cases to repeated cases. Effective cases refer to the test input that triggers at least one new statement or branch. The generation efficiency is described by the number of effective test cases generated per unit time, and the time statistics are accurate to the second level. In order to ensure the comparability of the experimental results, all experiments were repeated for 5 times, and the final results were evaluated by the arithmetic average, and the fluctuation range of coverage was controlled within 2%.

4.2 Experimental Results and Data Analysis

4.2.1 Analysis of the Number and Effectiveness of Automatically Generated Test Cases

The experimental results of automatic generation of test cases are analyzed from three dimensions: generation scale, effectiveness ratio and execution cost. To ensure the consistency of comparison basis, different methods complete the task of generating test cases under the same hardware environment and 60-minute running time limit, and count the final output results. Table 4 gives the comparative data of random test, traditional genetic algorithm, neural network-driven method and GA-NN model in the number and effectiveness of test cases.

Table 4. Number and effectiveness of test cases generated by different methods

Method name	Total number of generated use cases	Number of effective use cases	Invalid use case number	Proportion of effective use cases (%)	Average execution time (ms)
random test	1860	742	1118	39.89	12.4
Traditional genetic algorithm	1420	815	605	57.39	15.1
Neural network drive	1280	801	479	62.58	14.7
GA-NN model	1350	982	368	72.74	15.3

As shown in Table 4, from the scale of generation, the number of test cases generated by random testing per unit time is higher than that of other methods, and the number of invalid cases exceeds half of the total, reflecting that a large number of test inputs cannot reach the new program path when the input distribution lacks constraints. The traditional genetic algorithm has converged in the number of generations, but the proportion of effective use cases has been limited, and repeated input and similar paths still appear frequently. The neural network-driven method is further compressed in the generation scale, and the number of use cases is reduced to about 1280, and the effective ratio is increased to more than 62%, which shows that the prediction mechanism has formed a preliminary screening for low-value inputs. The GA-NN model maintains a more stable balance between generation scale and effectiveness.

The number of effective use cases is close to 1000, and the effective ratio is stable above 70%. The collaborative mechanism of genetic search and coverage prediction can obviously inhibit invalid input.

4.2.2 Analysis of Test Coverage Improvement Experiment Results

In the test coverage experiment, the accessibility of different methods to program sentence structure and branch structure is emphatically investigated. Coverage statistics are based on the execution results of the complete test set, excluding cases of midway failure or abnormal termination, and only counting successful execution samples. Table 5 shows the final results of each method in terms of statement coverage and branch coverage.

Table 5. Comparison of Test Coverage Results by Different Methods

Method name	Statement coverage (%)	Branch coverage (%)	Statement coverage growth	Branch coverage growth
random test	68.4	54.1	18.2	15.6
Traditional genetic algorithm	74.9	61.8	24.7	23.3
Neural network drive	77.3	64.2	27.1	25.7
GA-NN model	82.6	70.4	32.4	31.9

As shown in Figure 3 below, the improvement of coverage and the effectiveness of test cases show an obvious consistent trend. Random testing grew rapidly in the initial period of coverage, but the growth rate slowed down significantly in the middle and late period, and the coverage rate stayed in a low range. Traditional genetic algorithm is superior to random test in sentence coverage and branch coverage, which shows that search mechanism has a positive effect on path exploration. Neural network driving method improves the coverage level and the coverage growth is more concentrated, which shows that the prediction results guide the selection of test input. GA-NN model achieves the highest values in both types of coverage indicators, with sentence coverage exceeding 82% and branch coverage exceeding 70%.

The coverage growth rate is the most stable among all methods, and the coverage promotion is not concentrated in a single structure type, but keeps a balanced distribution among different control structures.

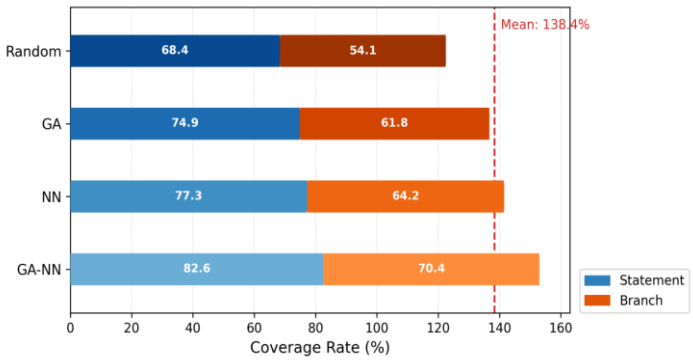

Figure 3: Comparison of coverage of different test methods

4.2.3 Analysis of Coverage Change Results During Model Iteration

The evolution characteristics of internal optimization process of GA-NN model are analyzed, and the changes of average coverage and maximum coverage of the model in continuous iteration process are recorded. The maximum number of iterations of genetic algorithm is set to 30 generations, and the coverage rate is unified after each generation is completed.

Table 6 shows the coverage change data of typical iterative nodes.

Table 6. Results of coverage change during model iteration

Iterative algebra	Average coverage (%)	Maximum coverage (%)	Single-generation growth rate
one	61.3	64.8	—
five	68.7	71.2	1.8
10	74.9	77.5	1.3
15	78.6	81.0	0.7
twenty	81.2	83.4	0.4
25	82.1	84.1	0.2
thirty	82.6	84.6	0.1

As shown in Figure 4 below, from the iterative trajectory, the coverage improvement shows obvious stage characteristics. The coverage rate of the first 10 generations increased rapidly, with an average increase of more than 1% per generation. The model quickly excluded the low coverage input in the initial stage. From the 10th generation to the 20th generation, the coverage rate continued to increase, but the growth rate gradually converged, and the test case search gradually concentrated in complex branch areas. After 20 generations, the change of coverage tends to be flat, and the growth rate of single generation is less than 0.5%.

The model enters a stable range and the search space is gradually saturated. The change process reflects that the cooperative relationship between prediction module and genetic optimization module is gradually stable, and the distribution of test input no longer fluctuates violently.

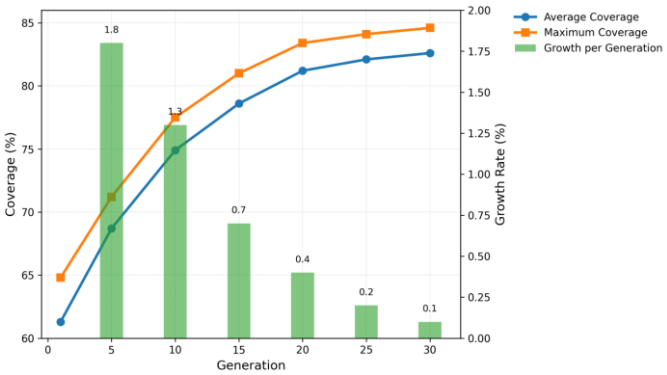

Figure 4: Variation trend of coverage with iteration times

4.2.4 Comparison of Comprehensive Experimental Results with Comparison Methods

In the comparison of comprehensive experimental results, three indicators, namely coverage level, effective test case size and generation efficiency, are introduced to systematically analyze the performance of different methods in the overall performance level.

All the methods completed the experiment under the same running time of 60 minutes and the same test object, and the statistical results are summarized in Table 7.

Table 7. Comparison Table of Comprehensive Performance Results of Different Test Methods

Method name	Final Statement Coverage (%)	Final Branch Coverage (%)	Number of effective use cases	Total number of use cases	Number of effective use cases per unit time (pieces /min)
random test	68.4	54.1	742	1860	12.4
Traditional genetic algorithm	74.9	61.8	815	1420	13.6
Neural network drive	77.3	64.2	801	1280	13.3
GA-NN model	82.6	70.4	982	1350	16.4

As shown in Figure 5 below, from the distribution of comprehensive indicators, random testing is dominant in the generation scale, but the proportion of effective use cases and coverage results are in the lowest range. Traditional genetic algorithm has improved obviously in coverage and effective scale, but the output per unit time is still limited by search efficiency. The coverage level of neural network-driven method is slightly higher than that of traditional genetic algorithm, but the growth rate of effective use cases is limited, indicating that a single prediction mechanism has constraints on the exploration of test space. GA-NN model is in the leading position in coverage, effective scale and generation efficiency, and the number of effective use cases per unit time reaches 16.4. The coverage improvement is not accompanied by scale expansion, and the utilization of test resources is more concentrated.

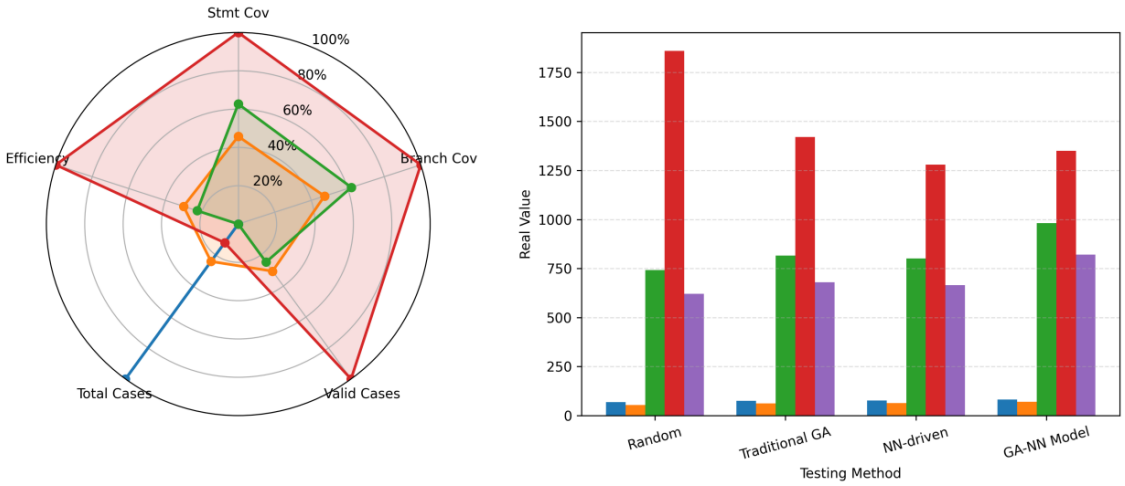

Figure 5: Comparison of comprehensive performance results of different test methods

4.2.5 Test Case Diversity and Redundancy Experimental Results Analysis

The diversity and redundancy of test cases reflect the distribution characteristics of test input in program path space.

This experiment makes statistical analysis from three dimensions: the number of different execution paths, the proportion of repeated use cases and the average path depth. The results are shown in Table 8.

Table 8. Statistics of Test Case Diversity and Redundancy Results

Method name	Number of different execution paths	Proportion of repeated use cases (%)	Average path depth	Deepest path length
random test	214	41.7	6.2	11
Traditional genetic algorithm	287	29.4	7.1	14
Neural network drive	301	26.8	7.5	15
GA-NN model	356	18.9	8.3	18

As shown in Table 8, the test inputs generated by random tests are concentrated in the shallow path area, with the repetition ratio exceeding 40%, and the path exploration scope is limited. The traditional genetic algorithm has improved the number and depth of paths, but the proportion of repeated use cases is still close to 30%. The neural network-driven method further expands the coverage of the path, and the repetition ratio decreases slightly, but the path depth is limited. GA-NN model is the most outstanding in diversity index.

The number of different execution paths reaches 356, and the repetition ratio is controlled within 20%. The reaching ability of deep paths is obviously enhanced, the distribution of test inputs is more dispersed, and the redundancy phenomenon is effectively suppressed.

4.2.6 Stability and Robustness Analysis of Experimental Results

To test the consistency of the performance of the model under different random initial conditions, the GA-NN model, the traditional genetic algorithm and the neural network-driven method were repeated five times independently.

Each experiment reinitializes the population and model parameters, and counts the distribution of final sentence coverage.

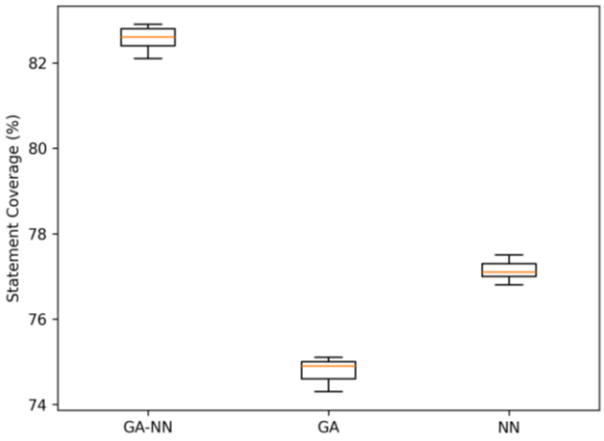

Figure 6: Box diagram of repeated experiment coverage fluctuation

As shown in Figure 6, from the coverage distribution interval, the gap between upper and lower quartiles of GA-NN model is the smallest, the coverage is concentrated in the range of 82.1% to 82.9%, and the fluctuation range is controlled within 0.4%. Traditional genetic algorithm and neural network driving method cover a wider distribution range and have obvious dispersion. The GA-NN model keeps stable output under different random conditions, the coverage results are insensitive to the initial state changes, and the overall operation process presents strong consistency.

4.3 Experimental Results Discussion and Model Validity Analysis

Combined with the above experimental results, different test methods have obvious differences in test case generation quality, coverage improvement range and operation stability. Random testing has an advantage in the number of generated tests, but the proportion of effective use cases is below 40% for a long time, and the coverage growth tends to stagnate in the middle and later stages, reflecting that when the input distribution lacks guidance, a lot of test resources are consumed in repeated paths.

The traditional genetic algorithm is superior to random test in coverage performance, the sentence coverage rate is stable in the range of 74% to 75%, and the proportion of repeated use cases is still close to 30%. The search process gradually converges to a local area in the middle and late stage, and the path expansion ability is limited. The neural network-driven method further improves the coverage index, and the branch coverage rate is close to 65%. The prediction mechanism forms certain constraints on input selection, but the scale of test cases shrinks obviously, and the growth of deep path accessibility is limited.

The performance of GA-NN model in many indicators shows consistent advantages. The statement coverage rate is kept above 82%, the branch coverage rate is about 70%, the effective use case ratio is over 70%, and the repeated use case ratio is controlled within 20%. The improvement of coverage does not depend on the significant expansion of test case scale, but focuses on the improvement of input quality. The growth curve of coverage shows that the model completed the main optimization in the first 10 generations, and the coverage improvement gradually narrowed after 15 generations, and entered a stable range after 20 generations, indicating that the search process gradually converged without violent fluctuations. In repeated experiments, the fluctuation range of the final coverage rate is controlled within 0.4%, the output results are concentrated and the running state is stable.

From the perspective of model structure, there is a clear division of labor between coverage behavior prediction and genetic search. The prediction module filters the low coverage input in advance to reduce invalid search, and the genetic search is responsible for maintaining diversity in the candidate space and promoting path expansion. The synergy between them inhibits the accumulation of repeated test input. Compared with the single method, the structure shows more balanced characteristics in terms of path number, path depth and coverage stability. It should be pointed out that the lifting speed of the model obviously slows down in the high coverage stage, and the growth rate of the coverage rate is less than 0.5% after 80%, and it is still difficult to reach the nested path with complex conditions. This phenomenon shows that the model is still affected by the complexity of program structure in deep path mining. On the whole, the experimental results support the effectiveness of the model in the scenario of automatic test case generation and coverage improvement, and its advantages are concentrated in the utilization efficiency of test resources and the stability of results.

5. Conclusions

5.1 Summary of Research Conclusions

Focusing on the automatic generation of software engineering test cases and the improvement of coverage, this paper constructs and verifies the test case generation model of genetic algorithm optimized neural network. The experimental results show that the model is superior to random test, traditional genetic algorithm and single neural network method in terms of test case effectiveness, coverage growth rate and operation stability under the same running time and test object conditions. The coverage rate of statements is kept in the range of 82% to 83%, the coverage rate of branches is kept around 70%, the proportion of effective test cases is over 70%, and the proportion of repeated cases is controlled within 20%. The growth process of coverage shows the characteristics of stages, and the main optimization is completed in the first 10 generations, and then it enters the convergence range after 20 generations, and the coverage fluctuation range is controlled within 0.4%. The experimental results show that the cooperative operation of coverage behavior prediction and genetic search can reduce invalid search and improve the quality of test input, and the utilization efficiency of test resources is obviously improved.

5.2 Model Innovation and Engineering Application Value

The model in this paper has formed a clear division of labor in structural design and operation mechanism. The coding method of test cases is consistent with the input parameters of the program to avoid illegal input interfering with the search process. Neural network undertakes the task of coverage behavior prediction, provides pre-constraints for test case screening, and genetic algorithm is responsible for maintaining search diversity and promoting path expansion, which form a closed-loop operation mechanism. Compared with the traditional coverage-driven method, this model does not rely on the expansion of test cases to achieve coverage improvement, but focuses on input distribution adjustment and path quality optimization. In engineering practice, the model is suitable for business system test scenarios with many interface parameters and complex control logic, and can be embedded in continuous integration pipeline to output high coverage test sets within a fixed time budget, providing stable support for version regression testing. The parameters of the model are clearly set, and the configuration of population size 50 and iteration upper limit 30 generations is stable in the experiment, which has direct migration value.

5.3 Research Deficiency and Future Improvement Direction

Although the model is stable in most experimental indexes, there is still room for improvement. When the coverage rate is above 80%, the increase rate obviously converges, and it is still difficult to reach the nested path with complex conditions, which shows that the prediction model has limited ability to identify the deep control structure. At present, sentence coverage and branch coverage are used as evaluation indexes in the experiment, while path coverage and abnormal coverage are not included in the analysis scope, so there is still room for expansion of evaluation dimensions. The test object focuses on medium-sized open source systems and has not covered micro-service architecture or high concurrency system scenarios. Subsequent research can introduce finer coverage indicators, combine with runtime feature information to enrich the prediction input features, verify the performance of the model in larger-scale engineering projects, and improve the adaptability of the model to complex software structures.

References

[1] Katoch S, Chauhan SS, Kumar V. A review on genetic algorithm: past, present, and future. Multimedia Tools Appl. 2021; 80(5):8091–8126. doi:10.1007/s11042-020-10139-6.

[2] Shi XH. Research on the optimisation of complex models of large-scale building structures dependent on adaptive grey genetic algorithms. Int J Biometrics. 2020; 12(1):13–28. doi:10.1504/IJBM.2020.105620.

[3] Chang H. Research on complex network community identification based on partially optimized genetic algorithm. J Comput Methods Sci Eng. 2020; 20(1):81–89. doi:10.3233/JCM-193649.

[4] Shi RF. Research on data mining system based on artificial intelligence and improved genetic algorithm. J Intell Fuzzy Syst. 2021; 40(4):6731–6742. doi:10.3233/JIFS-189507.

[5] Sun Y, Li JR, Fu XL, Wang HF, Li HH. Application research based on improved genetic algorithm in cloud task scheduling. J Intell Fuzzy Syst. 2020; 38(1):239–246. doi:10.3233/JIFS-179398.

[6] Wang XJ, Gan LS, Liu SL. Research on intelligence analysis technology of financial industry data based on genetic algorithm. J Supercomput. 2020; 76(5):3391–3401. doi:10.1007/s11227-018-2584-2.

[7] Zhou W, Zhao YJ, Chen WW, Liu YH, Yang RJ, Liu Z. Research on investment portfolio model based on neural network and genetic algorithm in big data era. Eurasip J Wirel Commun Netw. 2020; 2020(1):228. doi:10.1186/s13638-020-01850-x.

[8] Kotyrba M, Volna E, Habiballa H, Czyz J. The influence of genetic algorithms on learning possibilities of artificial neural networks. Computers. 2022; 11(5):70. doi:10.3390/computers11050070.

[9] Zheng L. Research on application of improved genetic algorithm and BP neural network in air quality evaluation. Fresenius Environ Bull. 2022; 31(6A):6043–6052.

[10] Yang JF. Indoor space compositions based on genetic algorithms to optimize neural networks. Phys Commun. 2020; 42:101167. doi:10.1016/j.phycom.2020.101167.

[11] Jiang CY. Research on the optimization of mortar mix proportion based on neural network models and genetic algorithm. Front Phys. 2025; 13:1557999. doi:10.3389/fphy.2025.1557999.

[12] Qiang Y, Xu XL, Li L, Liang SY, Wang X, Chen C, Tan XY. High-precision debris flow scale prediction model based on CIHHO algorithm combined with multilayer perceptron neural network. IEEE Access. 2024; 12:144710–144724. doi:10.1109/ACCESS.2024.3471795.

[13] Kim M. Software engineering for data analytics. IEEE Softw. 2020; 37(4):36–42. doi:10.1109/MS.2020.2985775.

[14] Sasmito AP, Kustono D, Purnomo P, Elmunsyah H, Nurhadi D, Sekarsari P. Development of Android-based teaching material in software engineering subjects for informatics engineering students. Int J Eng Pedagogy. 2021; 11(2):25–40. doi:10.3991/ijep.v11i2.16299.

[15] Mischke R, Schaffert K, Schneider D, Weinert A. Automated and manual testing in the development of the research software RCE. In: Groen D, DeMulatier C, Paszynski M, Krzhizhanovskaya VV, Dongarra JJ, Sloot PMA, editors. Computational Science – ICCS 2022. Lecture Notes in Computer Science. Vol 13353. Cham: Springer; 2022. p. 531–544. doi:10.1007/978-3-031-08760-8_44.

[16] Eisty NU, Carver JC. Testing research software: a survey. Empir Softw Eng. 2022; 27(6):138. doi:10.1007/s10664-022-10184-9.

[17] Zein S, Salleh N, Grundy J. Systematic reviews in mobile app software engineering: a tertiary study. Inf Softw Technol. 2023; 164:107323. doi:10.1016/j.infsof.2023.107323.

[18] Eisty NU, Kanewala U, Carver JC. Testing research software: an in-depth survey of practices, methods, and tools. Empir Softw Eng. 2025; 30(3):81. doi:10.1007/s10664-025-10620-6.

[19] Wang WB. Research on software regression testing method considering greedy algorithm sorting weight. Int J Bio-Inspired Comput. 2024; 24(1). doi:10.1504/IJBIC.2024.140130.

[20] Kaur G, Tandon A, Mittal R, Singh A. Reliability assessment of coverage execution oriented software growth model and genetic algorithm driven release planning. Int J Syst Assur Eng Manag. 2025. doi:10.1007/s13198-024-02632-0.

[21] Zheng LQ, Arasteh B, Mehrabani MN, Abania AV. An automatic software testing method to discover hard-to-detect faults using hybrid Olympiad optimization algorithm. J Electron Test. 2024; 40(4):539–556. doi:10.1007/s10836-024-06136-4.

[22] Alrawashdeh TA, ElQirem F, Althunibat A, Alsoub RA. A prioritization approach for regression test cases based on a revised genetic algorithm. Inf Technol Control. 2021; 50(3):443–457. doi:10.5755/j01.itc.50.3.27662.

[23] Bertolino A, Miranda B, Pietrantuono R, Russo S. Adaptive test case allocation, selection and generation using coverage spectrum and operational profile. IEEE Trans Softw Eng. 2021; 47(5):881–898. doi:10.1109/TSE.2019.2906187.

[24] Arasteh B, Arasteh K, Ghaffari A. An automatic software test-generation method to discover the faults using fusion of machine learning and horse herd algorithm. J Supercomput. 2025; 81(5):741. doi:10.1007/s11227-025-07219-5.