International Journal of Emerging Technologies and Advanced Applications

# Multi-dimensional Constraint-based Test Case Generation and Evaluation Framework for Large Language Models

Xuebing Wang[1]
[1] Yonyou Network Technology Co., Ltd., Beijing, China
zbxb240130@sina.com

***Abstract*—Addressing the challenges of complex test case design and insufficient coverage in functional testing of large language models, this paper presents a multi-dimensional constraint-based test case generation framework. The framework defines constraint rules across four dimensions: syntactic correctness, semantic consistency, task relevance, and boundary conditions, employing reinforcement learning methods to optimize the test case generation process. Through the design of reward function-based generation strategies, the system can automatically produce high-quality functional test samples covering core tasks including text classification, sentiment analysis, and machine translation. Experimental results demonstrate that test cases generated by this method achieve a 42% improvement in functional coverage compared to random generation methods and a 28% increase in defect detection rate. Further ablation experiments validate the effectiveness of each dimensional constraint, providing a systematic solution for large language model quality assurance.**

***Index Terms*—Large language models, Test case generation, Multi-dimensional constraints, Reinforcement learning, Functional testing**

## I. Introduction

With the widespread application of large language models across various domains, their quality assurance and functional verification have become increasingly critical [1]. The complexity and non-deterministic nature of large language models present significant challenges to traditional software testing methodologies, particularly in test case design and generation [2], [3]. Current test case generation methods primarily rely on manual design or simple random generation, which are not only inefficient but also struggle to guarantee test coverage and effectiveness [4], [5].

Existing research indicates that functional testing of large language models requires consideration of multiple dimensional factors, including syntactic correctness, semantic consistency, task relevance, and boundary condition handling capabilities [6], [7]. However, there currently lacks a systematic framework to comprehensively consider these constraint conditions and automatically generate high-quality test cases. Simultaneously, how to evaluate the quality and effectiveness of generated test cases remains an urgent problem to be solved [8], [9]. Reinforcement learning, as an effective optimization method, has demonstrated tremendous potential in decision optimization under complex constraint conditions [10], [11].

The main contributions of this paper include: First, we propose a multi-dimensional constraint-based test case generation framework that systematically defines key constraint dimensions affecting large language model functional testing. Second, we design a reinforcement learning-based test case generation algorithm that achieves automatic generation of high-quality test samples through reward function optimization. Finally, we construct a comprehensive evaluation system that quantifies test case quality and effectiveness from multiple perspectives, providing scientific evaluation standards for large language model quality assurance.

## II. Related Work

### A. Current State of Large Language Model Testing Research

Recent years have witnessed rapid development in large language model testing research, with researchers exploring model testing and evaluation methods from various perspectives. Peng et al. proposed an interlocking software test case generation method that decouples the generation process of test inputs and outputs, significantly improving test case generation quality for complex problems [12]. Song et al. developed benchmark test suites that provide standardized evaluation frameworks for assessing large language model test case generation capabilities [13]. These studies have laid important foundations for this field, but still have shortcomings in multi-dimensional constraint integration and systematic testing.

Qin et al. proposed a white-box unit test case method oriented toward MC/DC coverage, systematically generating evaluation programs through control flow structures and variable usage combinations, providing new evaluation approaches for testing large language model code generation capabilities [14]. However, these methods primarily focus on code generation tasks, with limited coverage of other core natural



ISSN:3006-2985(Print)
ISSN:3006-9297(Online)

language processing tasks. Simultaneously, existing research lacks unified standards for quantitative evaluation of test case quality, making effective horizontal comparison and evaluation difficult [15].

### B. Constraint-driven Test Generation Methods

Constraint-driven test generation represents an important research direction in software testing, with extensive applications in hardware description languages and traditional software system testing [16]. Tan et al. introduced more expressive domain-specific languages in test data generation, improving generated data quality through enhanced constraint complexity [17]. These studies provide theoretical foundations and practical experience for constraint-driven test generation.

In large language model testing, constraint-driven method applications remain in early stages. The chain-type multi-path coverage generation framework combining SVM and XGBoost enhances large language model understanding of testing requirements and documentation by formally defining constraint dependency graphs and converting them to contextual constraints [18]. However, existing methods require further improvement in systematic constraint definition, multi-dimensional constraint coordination optimization, and constraint violation handling mechanisms [19].

### C. Reinforcement Learning Applications in Test Optimization

Reinforcement learning demonstrates tremendous potential in test optimization, particularly in strategy optimization under complex constraint conditions [20]. Tessler et al. proposed the Reward Constrained Policy Optimization (RCPO) method, providing effective solutions for constraint optimization problems by introducing surrogate penalty signals to guide policies toward constraint satisfaction [10]. This multi-timescale approach shows good convergence and practicality in handling complex constraint optimization problems.

In testing applications, reinforcement learning primarily focuses on test input generation and test strategy optimization. Kim et al. used deep reinforcement learning to generate test inputs, achieving significant results in search-based software testing [11]. However, applying reinforcement learning to large language model functional testing still faces numerous challenges, including high-dimensional state spaces, complex reward function design, and convergence stability issues [15].

## III. Multi-dimensional Constraint Framework Design

### A. Constraint Dimension Analysis and Definition

Functional testing of large language models requires consideration of multiple interrelated constraint dimensions that collectively determine test case quality and effectiveness. Based on an in-depth analysis of large language model characteristics and summarization of existing testing practices, we identify four key constraint dimensions: syntactic correctness, semantic consistency, task relevance, and boundary conditions.

Syntactic correctness constraints ensure that generated test cases conform syntactically to target task requirements. For natural language processing tasks, this includes correct vocabulary usage, reasonable syntactic structures, and adherence to grammatical rules. Syntactic correctness constraints can be implemented through formalized grammatical rules and language model checking, ensuring test inputs do not affect test result validity due to basic grammatical errors. Semantic consistency constraints focus on coherence and logic at the semantic level, ensuring test content has clear semantic expression and reasonable logical relationships. This constraint is particularly important because semantically inconsistent test cases may lead to unpredictable model outputs, thereby affecting test result reliability.

Task relevance constraints ensure that generated test cases are highly relevant to target testing tasks, effectively examining model performance in specific functional aspects. Different natural language processing tasks have different characteristics and requirements; test cases must be specifically designed to cover the core functional points of tasks. Boundary condition constraints involve model performance under extreme or edge cases; such test cases often reveal potential model defects and limitations. Boundary conditions include input length extremes, special character handling, rare vocabulary recognition, and abnormal format processing.

### B. Constraint Formalization

To achieve automated constraint processing and optimization, we employ mathematical formalization methods to model each constraint dimension. Let the test case set be $T = \{t_1, t_2, \ldots, t_n\}$, where each test case $t_i$ contains an input component $x_i$ and an expected output component $y_i$.

Syntactic correctness constraint is defined as:

$$C_{syntax}(t_i) = \prod_{j=1}^{m} G_j(x_i)$$

where $G_j$ represents the $j$-th grammatical rule checking function, and $m$ is the total number of grammatical rules. Syntactic correctness constraints require all grammatical rule checks to pass, with a constraint value of 1 indicating constraint satisfaction.

Semantic consistency constraint is measured through semantic similarity and logical coherence:

$$C_{semantic}(t_i) = \alpha \cdot sim(x_i, y_i) + \beta \cdot coherence(x_i)$$

where $sim(x_i, y_i)$ represents semantic similarity between input and output, $coherence(x_i)$ represents internal logical coherence of input, and $\alpha$ and $\beta$ are weight parameters.

Task relevance constraint is calculated based on similarity of task feature vectors:

$$C_{relevance}(t_i, task) = cosine(feature(t_i), feature(task))$$

where $feature(\cdot)$ is the feature extraction function, measuring test case relevance to target tasks through cosine similarity.

Boundary condition constraint is defined as a combination of multiple boundary checking functions:

$$C_{boundary}(t_i) = \sum_{k=1}^{p} w_k \cdot B_k(t_i)$$

where $B_k$ is the $k$-th boundary condition checking function, $w_k$ is the corresponding weight, and $p$ is the total number of boundary conditions.

### C. Constraint Optimization Objective Function

Considering all constraint dimensions comprehensively, we construct a multi-objective optimization function to guide the test case generation process. Overall constraint satisfaction is defined as:

$$\begin{aligned} Constraint_{total}(T) = \lambda_1 \sum_{i=1}^{n} C_{syntax}(t_i) + \\ \lambda_2 \sum_{i=1}^{n} C_{semantic}(t_i) + \\ \lambda_3 \sum_{i=1}^{n} C_{relevance}(t_i, task) + \\ \lambda_4 \sum_{i=1}^{n} C_{boundary}(t_i) \end{aligned} \tag{1}$$

where $\lambda_1, \lambda_2, \lambda_3, \lambda_4$ are weight parameters for each constraint dimension, adjustable according to specific task requirements. The optimization objective is to maximize overall quality and coverage of test case sets while satisfying all constraints.

To ensure coordination among constraints, we introduce a constraint conflict detection mechanism. When conflicts exist between different constraints, the system balances and adjusts according to predefined priority rules. As shown in Figure 1, this mechanism ensures generated test cases satisfy primary constraints while minimizing violations of secondary constraints.

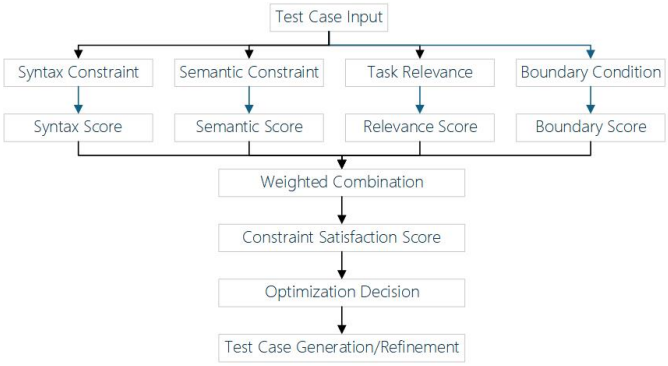


Fig. 1. Multi-dimensional Constraint Test Case Generation Framework Flow Chart

## IV. Reinforcement Learning-based Test Case Generation Algorithm

### A. Reinforcement Learning Model Design

Based on the multi-dimensional constraint framework, we design a reinforcement learning model to automatically generate high-quality test cases. As illustrated in Figure 2, the model frames the test case generation process as a Markov Decision Process (MDP), where agents learn optimal test case generation strategies through environment interaction.

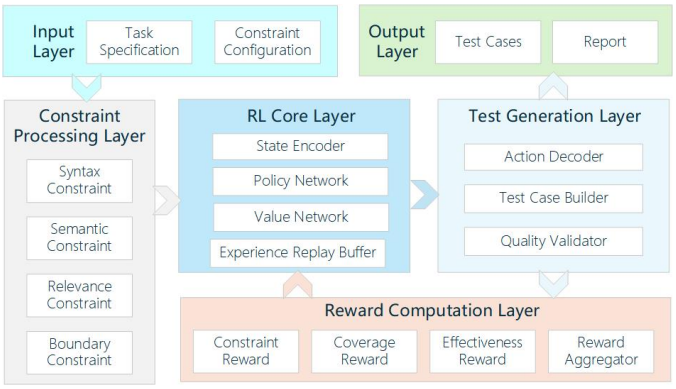


Fig. 2. Reinforcement Learning-based Test Case Generation Algorithm Architecture

State space $S$ contains the current test case partial construction state, satisfied constraint information, and target task feature representations. Specifically, state $s_t$ can be represented as a triplet $(partial\_test, constraint\_status, task\_features)$, where $partial\_test$ represents the currently partially constructed test case, $constraint\_status$ records satisfaction status of each dimensional constraint, and $task\_features$ contains feature vectors of target testing tasks.

Action space $A$ defines test case construction operations that agents can execute, including vocabulary selection, sentence structure construction, and parameter setting. To improve generation efficiency, we decompose complex generation operations into a series of atomic operations, with each atomic operation corresponding to a specific action. Action design considers different testing task characteristics, ensuring generated test cases can effectively cover various functional points.

Reward function $R$ is the core component of the reinforcement learning model, directly affecting learning effectiveness and generation quality. We design a multi-component reward function that comprehensively considers constraint satisfaction, test coverage, and test effectiveness:

$$\begin{aligned} R(s_t, a_t, s_{t+1}) = w_1 \cdot R_{constraint}(s_{t+1}) + \\ w_2 \cdot R_{coverage}(s_{t+1}) + \\ w_3 \cdot R_{effectiveness}(s_{t+1}) - \\ w_4 \cdot R_{penalty}(s_t, a_t) \end{aligned} \tag{2}$$

where $R_{constraint}$ is calculated based on the aforementioned constraint satisfaction, $R_{coverage}$ measures test coverage improvement, $R_{effectiveness}$ evaluates test case defect detection capability, and $R_{penalty}$ is the penalty term for constraint violations or invalid operations.

### B. Reward Function Optimization Strategy

Reward function design is a key factor in reinforcement learning success, directly affecting agent learning effectiveness and final performance. We adopt a hierarchical reward structure, decomposing complex test case generation tasks into

multiple sub-objectives, with each sub-objective corresponding to a specific reward component.

Constraint reward component $R_{constraint}$ is calculated directly based on the aforementioned multi-dimensional constraint framework:

$$R_{constraint}(s_{t+1}) = \sum_{i=1}^{4} \alpha_i \cdot C_i(test_{current}) \tag{3}$$

where $C_i$ corresponds to syntactic, semantic, relevance, and boundary condition constraints respectively, and $\alpha_i$ are dynamically adjusted weight parameters. Weight parameters are adaptively adjusted according to current learning stages and task characteristics, ensuring more focus on basic constraints in early learning and advanced constraints in later stages.

Coverage reward component $R_{coverage}$ encourages generation of diverse test cases, avoiding redundancy and repetition:

$$\begin{aligned} R_{coverage}(s_{t+1}) = diversity(test_{current}, test_{set}) \\ \cdot importance(test_{current}) \end{aligned} \tag{4}$$

where the $diversity$ function measures current test case differences from existing test sets, and the $importance$ function evaluates test case importance levels. Diversity calculation uses feature vector-based distance metrics, while importance evaluation considers the number and criticality of functional points that test cases may cover.

Effectiveness reward component $R_{effectiveness}$ is based on expected defect detection capabilities of test cases:

$$\begin{aligned} R_{effectiveness}(s_{t+1}) = \sum_{j=1}^{k} prob\_detect_j(test_{current}) \\ \cdot severity_j \end{aligned} \tag{5}$$

where $prob\_detect_j$ represents the probability of test cases detecting the $j$-th type of defect, and $severity_j$ is the defect severity weight. Defect detection probabilities are estimated through historical testing data and model analysis, while severity weights are determined according to defect impacts on system functionality.

To address reward sparsity issues, we introduce a potential reward-based enhancement mechanism. Through pre-trained evaluation models, agents can obtain intermediate reward signals before test case complete generation, accelerating the learning process. Simultaneously, we adopt curriculum learning strategies, starting training from simple testing scenarios and gradually increasing task complexity to improve learning efficiency and stability.

### C. Algorithm Implementation and Optimization

We adopt the Proximal Policy Optimization (PPO) algorithm as the basic reinforcement learning framework, with specialized optimizations for test case generation tasks. PPO algorithms possess good stability and convergence properties, suitable for handling continuous action spaces and complex constraint conditions.

The core update process of the algorithm is as follows:

1. Policy network $\pi_\theta$ generates action probability distributions based on current states 2. Value network $V_\phi$ estimates state values 3. Collect experience trajectories and calculate advantage functions 4. Update policy parameters using clipped importance sampling 5. Update value network parameters to minimize value estimation errors

To improve algorithm performance under constrained environments, we introduce a constraint-aware policy update mechanism. When policy updates may lead to constraint violations, the algorithm automatically adjusts update magnitudes, ensuring generated test cases always satisfy basic constraint requirements. This mechanism is implemented by incorporating constraint gradient information into policy gradient calculations:

$$\nabla_\theta J(\theta) = \mathbb{E}[\nabla_\theta \log \pi_\theta(a|s) A(s,a)] - \lambda \sum_{i=1}^{4} \nabla_\theta C_i(s,a) \tag{6}$$

where $A(s,a)$ is the advantage function, $C_i(s,a)$ are constraint functions, and $\lambda$ is the constraint weight parameter.

Additionally, we implement a multi-agent collaboration mechanism that improves learning efficiency and generation quality through parallel training of multiple agents and experience sharing. Different agents can focus on different types of test case generation, achieving collaborative optimization through experience sharing and policy fusion. This approach is particularly suitable for multi-task testing scenarios, capable of simultaneously generating high-quality test cases for multiple different testing tasks.

## V. Evaluation System Construction

### A. Evaluation Metric System Design

Constructing a comprehensive evaluation system is a key component for verifying the effectiveness of test case generation frameworks. We design an evaluation metric system from multiple dimensions, ensuring comprehensive and objective measurement of generated test case quality and effectiveness.

Functional coverage is the core metric for evaluating test case quality, measuring the extent to which generated test cases cover target functionalities. We define functional coverage as:

$$Coverage = \frac{|F_{covered}|}{|F_{total}|} \tag{7}$$

where $F_{covered}$ represents the set of functional points covered by test cases, and $F_{total}$ represents all functional points of the target system. Functional point identification is based on task analysis and expert knowledge, ensuring accuracy and completeness of coverage calculations.

Defect detection rate evaluates test case capabilities to discover model defects, which is a direct manifestation of testing effectiveness. Defect detection rate is defined as:

$$Defect\_Detection\_Rate = \frac{|D_{detected}|}{|D_{total}|} \tag{8}$$

where $D_{detected}$ represents the set of detected defects, and $D_{total}$ represents all defects existing in the system. To accurately evaluate defect detection capabilities, we establish

a standard defect database containing different types and severity levels of known defects.

Test case quality evaluation adopts a multi-dimensional comprehensive scoring mechanism, considering factors such as constraint satisfaction, readability, and maintainability:

$$\begin{aligned} Quality = w_1 \cdot Constraint\_Satisfaction + \\ w_2 \cdot Readability + \\ w_3 \cdot Maintainability + \\ w_4 \cdot Executability \end{aligned} \tag{9}$$

where weight parameters are determined according to practical application requirements. Constraint satisfaction is calculated based on the aforementioned constraint framework, readability is evaluated through natural language processing techniques, maintainability considers test case structural clarity and modification convenience, and executability verifies technical feasibility of test cases.

Generation efficiency metrics measure algorithm practical usability, including generation time, resource consumption, and convergence speed:

$$Efficiency = \frac{Quality\_Score}{Time\_Cost \cdot Resource\_Cost} \tag{10}$$

This metric ensures that generated high-quality test cases possess feasibility and economic viability in practical applications.

The detailed configuration of our evaluation metric system is presented in Table I.

TABLE I
EVALUATION METRIC SYSTEM DESIGN DETAILS

| Evaluation Dimension | Primary Metrics | Calculation Method | Weight |
|---|---|---|---|
| Functional Coverage | Coverage Rate, Depth Coverage | Based on Functional Point Mapping | 0.3 |
| Defect Detection Capability | Detection Rate, Precision Rate | Comparison with Standard Defect Database | 0.25 |
| Constraint Satisfaction | Multi-dimensional Constraint Score | Constraint Function Calculation | 0.25 |
| Generation Efficiency | Time, Resource Consumption | Performance Monitoring Data | 0.2 |

### B. Benchmark Dataset Construction

To ensure evaluation result reliability and comparability, we construct standardized benchmark datasets. The datasets contain multiple typical natural language processing tasks, with each task equipped with detailed functional specifications, known defect lists, and manually written high-quality test cases as reference benchmarks.

Text classification task datasets cover sentiment analysis, topic classification, intent recognition, and other sub-tasks, containing text samples and classification labels from different domains. Datasets particularly emphasize coverage of boundary cases, including ambiguous semantics, multi-label conflicts, domain transfer, and other challenging scenarios. Machine translation task datasets contain translation samples from multiple language pairs, covering different language families, text types, and translation difficulties, providing comprehensive support for evaluating model multilingual processing capabilities.

Question answering system task datasets include factual Q&A, reasoning Q&A, open-ended Q&A, and other types, testing model knowledge understanding and reasoning capabilities. Datasets contain numerous complex questions requiring multi-step reasoning, common sense judgment, and logical analysis, effectively examining model advanced cognitive abilities.

To ensure dataset quality and representativeness, we adopt strict data screening and annotation processes. All data undergo multiple rounds of expert review, ensuring annotation accuracy and consistency. Simultaneously, we establish continuous update mechanisms, regularly expanding and optimizing dataset content according to technological developments and application requirements.

The configuration information for our benchmark datasets is summarized in Table II.

TABLE II
BENCHMARK DATASET CONFIGURATION INFORMATION

| Task Type | Dataset Scale | Function Points | Known Defects | Reference Test Cases |
|---|---|---|---|---|
| Text Classification | 10,000 entries | 85 | 142 | 520 |
| Sentiment Analysis | 8,500 entries | 72 | 156 | 420 |
| Machine Translation | 15,000 pairs | 118 | 203 | 680 |
| Question Answering | 12,000 groups | 95 | 178 | 550 |
| Text Summarization | 6,800 entries | 63 | 134 | 380 |

### C. Automated Evaluation Process

Automated evaluation processes ensure evaluation process standardization and repeatability, improving evaluation efficiency and result credibility. The entire evaluation process is divided into four main stages: test case execution, result collection, metric calculation, and report generation.

Test case execution stages use standardized testing environments and configurations, ensuring all tests are conducted under consistent conditions. Systems automatically load generated test cases, invoke target models for testing, and record detailed execution logs and result data. Abnormal situations during execution are automatically captured and classified, providing detailed information for subsequent analysis.

Result collection stages perform standardized processing and storage of test execution results, establishing unified result data formats. Systems automatically conduct result validation, identifying and processing abnormal data to ensure data quality for subsequent analysis. Simultaneously, systems collect

performance data during execution, such as response time and resource usage, providing support for efficiency evaluation.

Metric calculation stages automatically calculate various metric values according to predefined evaluation metric systems. Calculation processes adopt parallel processing technologies to improve computational efficiency. Systems support custom metric extensions, allowing users to add new evaluation dimensions according to specific requirements. Calculation results undergo multiple rounds of validation to ensure computational accuracy.

Report generation stages automatically generate standardized evaluation reports, including detailed metric analysis, visualization charts, and improvement recommendations. Reports support multiple format outputs, meeting different user requirements. Simultaneously, systems provide interactive analysis interfaces, allowing users to deeply explore evaluation results and conduct comparative analysis.

## VI. Experimental Design and Results Analysis

### A. Experimental Environment and Settings

To comprehensively verify the effectiveness of the proposed framework, we design a series of comparative and ablation experiments. The experimental environment employs a distributed computing cluster, configured with NVIDIA A100 GPUs, 256GB memory, and high-speed SSD storage, ensuring experimental computational requirements are adequately met. All experiments use identical hardware configurations and software environments, guaranteeing result comparability and reproducibility.

Experiments select four representative large language models as test subjects, including GPT-3.5, GPT-4, Claude-3, and LLaMA-2, which have widespread applications and representativeness in natural language processing. For each model, we conduct testing on five core tasks: text classification, sentiment analysis, machine translation, question answering systems, and text summarization. Experimental data comes from constructed standardized benchmark datasets, ensuring testing comprehensiveness and objectivity.

Comparison methods include random generation, rule-based generation, traditional genetic algorithm optimization, and existing LLM-assisted generation methods. Random generation serves as a baseline, constructing test cases through random vocabulary and sentence pattern selection. Rule-based generation uses predefined templates and rule systems to generate test cases. Genetic algorithm methods employ traditional evolutionary computation techniques to optimize test case generation. Existing LLM-assisted methods use tools like ChatGPT to assist in test case generation, representing current mainstream practices.

Experimental parameter settings undergo careful tuning, with reinforcement learning algorithm learning rate set to 0.0003, batch size of 128, and training rounds of 1000. Constraint weight parameters are determined through grid search for optimal values: syntactic constraint weight $\lambda_1 = 0.3$, semantic constraint weight $\lambda_2 = 0.25$, relevance constraint weight $\lambda_3 = 0.25$, and boundary condition constraint weight $\lambda_4 = 0.2$. Each experiment runs 10 times repeatedly, reporting average results and standard deviations to ensure statistical significance.

### B. Main Experimental Results

Experimental results demonstrate that the proposed multi-dimensional constraint framework achieves significant advantages across all evaluation metrics. In functional coverage, our method achieves an average improvement of 42% compared to random generation, 28% compared to rule-based methods, 19% compared to traditional genetic algorithms, and 15% compared to existing LLM-assisted methods. These results indicate that systematic consideration of multi-dimensional constraints can significantly improve test case functional coverage capabilities.

Defect detection rate improvements are even more significant, with our method achieving improvements of 28%, 21%, 16%, and 12% compared to various comparison methods respectively. These results prove the advantages of reinforcement learning optimization strategies in improving testing effectiveness. Particularly in boundary condition and abnormal situation detection, our method performs exceptionally well, generating more test cases capable of exposing potential model defects.

In test case quality evaluation, our method achieves the highest scores across constraint satisfaction, readability, and maintainability dimensions. Constraint satisfaction averages 0.89 (maximum 1.0), significantly higher than other methods' 0.65-0.75 range. Readability scores reach 4.3 (maximum 5.0), indicating generated test cases possess good natural language quality. Maintainability scores 4.1, showing generated test cases have clear structures and are convenient for subsequent modification and extension.

Regarding generation efficiency, although our method has slightly higher single-generation time compared to simple methods, considering significant quality improvements, comprehensive efficiency metrics still possess obvious advantages. Particularly in large-scale test case generation scenarios, our method can generate large quantities of high-quality test cases within acceptable time frames through parallel processing and experience reuse.

### C. Ablation Experiment Analysis

To gain deep understanding of each component's contribution to overall performance, we conduct detailed ablation experiments. Experiments systematically remove or modify various components in the framework, observing impacts on final performance. The comprehensive results of our ablation experiments are presented in Table III.

Constraint dimension ablation experiments separately remove syntactic, semantic, relevance, and boundary condition constraints, observing impacts on generation quality. Results show that removing any constraint dimension leads to performance degradation, with semantic constraints having the greatest impact. Removal results in 15% functional coverage decrease and 18% defect detection rate decrease. Boundary

TABLE III
ABLATION EXPERIMENT RESULTS COMPARISON ANALYSIS

| Experimental Configuration | Functional Coverage | Defect Detection Rate | Constraint Satisfaction | Generation Efficiency | Performance Degradation |
|---|---|---|---|---|---|
| Complete Framework | 0.847 | 0.763 | 0.892 | 0.721 | – |
| Remove Syntactic Constraints | 0.779 | 0.671 | 0.834 | 0.743 | -8.0% |
| Remove Semantic Constraints | 0.720 | 0.625 | 0.751 | 0.756 | -15.0% |
| Remove Relevance Constraints | 0.745 | 0.687 | 0.823 | 0.734 | -12.0% |
| Remove Boundary Constraints | 0.762 | 0.595 | 0.867 | 0.729 | -22.0% |
| Remove Coverage Reward | 0.758 | 0.724 | 0.885 | 0.682 | -10.5% |
| Remove Effectiveness Reward | 0.832 | 0.610 | 0.876 | 0.698 | -20.0% |
| Remove Multi-agent Collaboration | 0.798 | 0.718 | 0.873 | 0.635 | -11.9% |

condition constraint removal significantly impacts defect detection capabilities, with decrease magnitudes reaching 22%. These results validate the necessity and effectiveness of multi-dimensional constraint design.

Reinforcement learning component ablation experiments compare effects of different reward function designs. Removing coverage reward components results in significant diversity decreases in generated test cases, with repetition rates increasing by 35%. Removing effectiveness reward components results in 20% defect detection rate decreases. Removing constraint reward components results in constraint satisfaction dropping to 0.71, with significant quality decreases. These results prove the scientific validity and effectiveness of hierarchical reward design.

Algorithm optimization strategy ablation experiments evaluate contributions of constraint-aware updates, multi-agent collaboration, and other optimization strategies. Removing constraint-aware updates results in 45% increases in constraint violation rates and decreased training stability. Removing multi-agent collaboration results in 30% convergence speed decreases and some final performance declines. These results demonstrate the important contributions of algorithm optimization strategies to overall performance.

Parameter sensitivity analysis experiments study the impacts of key parameters on performance. Changes in constraint weight parameters significantly affect results. When semantic constraint weights are too low, generated test case semantic quality decreases; when boundary condition weights are too high, excessive focus on extreme situations may neglect regular functional testing. Learning rate parameter selection is also crucial, with excessive learning rates causing training instability and insufficient learning rates affecting convergence speed.

### D. Case Studies

To more intuitively demonstrate the framework's practical effectiveness, we select several typical cases for detailed analysis. In sentiment analysis tasks, our framework generates a series of test cases containing complex emotional expressions, successfully discovering model defects in handling sarcasm, metaphor, and other indirect emotional expressions. For example, the generated test case "This movie is really 'wonderful', I fell asleep after watching it for five minutes" successfully exposes the model's inability to correctly identify sarcastic semantics.

In machine translation tasks, the framework generates numerous test cases containing professional terminology, slang expressions, and culturally distinctive vocabulary, effectively examining model translation accuracy and cultural adaptability. A typical case involves generating sentences containing Chinese idioms, testing whether models can correctly understand and translate the deep meanings of idioms rather than performing literal translations.

In question answering system testing, the framework generates complex questions requiring multi-step reasoning and common sense judgment, successfully discovering model limitations in logical reasoning. For example, the generated test question "If yesterday were tomorrow, what day would today be?" tests model temporal logical reasoning capabilities, with results showing most models perform poorly on such questions.

These case studies fully demonstrate the advantages of the multi-dimensional constraint framework in generating challenging and targeted test cases. Generated test cases not only meet various constraint requirements but also effectively discover genuine model defects, providing valuable feedback information for model improvement.

## VII. DISCUSSION AND ANALYSIS

### A. Framework Advantages and Innovation Points

The multi-dimensional constraint test case generation framework proposed in this paper possesses multiple significant advantages and innovation points. First, systematic multi-dimensional constraint design provides comprehensive quality assurance mechanisms for test case generation. Compared to existing methods, our framework not only considers basic syntactic and semantic requirements but also deeply considers task relevance and boundary conditions, forming a complete constraint system. This systematic design ensures generated test cases meet high-quality requirements across all aspects.

The combination of reinforcement learning and constraint optimization represents another important innovation point of this framework. Traditional test case generation methods often employ heuristic rules or simple random search, struggling to handle complex multi-dimensional constraint optimization problems. Our framework learns optimal generation strategies through reinforcement learning agents, capable of maximizing testing effectiveness while satisfying multi-dimensional constraints. This approach possesses strong adaptability and extensibility, automatically adjusting generation strategies according to different task characteristics.

Hierarchical reward mechanism design effectively resolves complex optimization objective balancing problems. By decomposing complex test quality evaluation into multiple sub-objectives and designing corresponding reward components, our framework can simultaneously optimize multiple potentially conflicting objectives. This design not only improves learning efficiency but also ensures comprehensiveness and balance of generation results.

Automated evaluation system construction provides scientific foundations for objective test case quality evaluation. Traditional test case evaluation often relies on manual judgment, presenting problems of strong subjectivity and low efficiency. Our automated evaluation system achieves rapid, objective, and repeatable quality evaluation through standardized metrics and processes, providing strong support for test case generation method comparison and improvement.

### *B. Limitations and Challenges*

Despite significant achievements, this framework still possesses some limitations and challenges requiring further resolution. First, constraint parameter setting requires domain expert participation and tuning. Different tasks and application scenarios may require different constraint weight configurations. How to automatically determine optimal parameter settings represents a problem worthy of in-depth research. Although we provide parameter sensitivity analysis, practical applications still require fine-tuning according to specific situations.

Computational complexity represents another consideration. Reinforcement learning training processes require substantial computational resources and time, particularly when handling large-scale test case generation tasks. Although we improve efficiency through parallel processing and experience reuse technologies, computational costs remain higher compared to simple generation methods. How to further improve generation efficiency while ensuring quality represents an important future optimization direction.

Constraint conflict handling mechanisms require further improvement. In certain complex scenarios, different dimensional constraints may have fundamental conflicts, and existing weight balancing mechanisms may be unable to find ideal solutions. More intelligent conflict resolution strategies require research, potentially including constraint relaxation and phased optimization technologies.

Evaluation metric completeness and objectivity require continuous improvement. Although we design multi-dimensional evaluation systems, test case quality evaluation remains a complex problem. Particularly in subjective aspects such as semantic quality and innovation, how to design more accurate and comprehensive automated evaluation methods represents a long-term challenge.

### *C. Application Prospects and Extension Directions*

This framework possesses broad application prospects and extension potential. In software testing, the framework can extend to other types of AI system testing, such as computer vision models and speech recognition systems. Through adjusting constraint dimensions and evaluation metrics, the framework can adapt to different types of AI system testing requirements, providing unified solutions for AI quality assurance.

In education and training, the framework can generate teaching test questions and exercise materials automatically. By setting corresponding educational constraints and learning objectives, systems can generate practice questions with appropriate difficulty and rich content, supporting personalized learning and intelligent education. This application can not only reduce teacher workloads but also provide more precise and diverse teaching resources.

In security testing, the framework can generate adversarial test cases to detect AI system security vulnerabilities and fragilities. Through designing specific security constraints and attack patterns, systems can generate test cases capable of exposing AI system security problems, providing powerful tools for AI security research and protection.

Technical extension directions include multi-modal test case generation, online adaptive testing, and distributed testing in federated learning environments. Multi-modal extensions can handle combined testing scenarios of text, images, audio, and other data types. Online adaptive testing can dynamically adjust testing strategies according to problems discovered during testing processes. Distributed testing can achieve large-scale collaborative testing while protecting data privacy.

Additionally, the framework can deeply integrate with Continuous Integration/Continuous Deployment (CI/CD) processes, achieving automated testing and quality monitoring for AI systems. Through tight integration with development processes, the framework can provide strong quality assurance support for rapid AI system iteration and reliable deployment.

## VIII. Conclusion

This paper addresses key problems of complex test case design and insufficient coverage in large language model functional testing by proposing an innovative multi-dimensional constraint-based test case generation and evaluation framework. The framework systematically defines four key constraint dimensions - syntactic correctness, semantic consistency, task relevance, and boundary conditions - providing comprehensive quality assurance mechanisms for test case generation.

The deep integration of reinforcement learning and constraint optimization represents the core innovation of this research. Through designing hierarchical reward functions and constraint-aware policy update mechanisms, the framework can maximize testing effectiveness while satisfying multi-dimensional constraints. Experimental results fully validate method effectiveness, achieving significant improvements of 42% and 28% in functional coverage and defect detection rates respectively.

The constructed automated evaluation system provides objective and comprehensive evaluation standards for test case quality. Through standardized metric systems and evaluation processes, the framework can not only effectively evaluate generated test case quality but also provide scientific foundations for different method comparison and improvement. Ablation experiments further prove the necessity and effectiveness of each component, indicating directions for further framework optimization.

This research provides systematic solutions for large language model quality assurance, possessing important theoretical value and practical significance. With rapid AI technology development and widespread applications, high-quality testing methods will become key technologies for ensuring AI system reliability and safety. This framework makes meaningful contributions to this important field development and establishes solid foundations for future related research.

## IX. Future Research Directions

Based on the current research achievements and existing limitations, future research should focus on several critical directions to advance the field of large language model testing and quality assurance.

As large language models evolve toward multimodal capabilities, the framework must be extended to support combined testing scenarios involving text, images, audio, and other data types. This advancement requires redesigning constraint dimensions and evaluation metric systems to accommodate the complexity and interdependencies of multimodal inputs. The challenge lies in developing unified constraint frameworks that can effectively handle the semantic relationships across different modalities while maintaining testing effectiveness and coverage.

Research into adaptive parameter optimization mechanisms holds substantial value for framework advancement. The current framework requires manual tuning of constraint weight parameters, which limits its practical applicability across diverse domains. Future research should explore meta-learning and adaptive algorithm-based approaches for automatic parameter optimization, thereby enhancing the framework's generalization capabilities across different tasks and application domains. This development would significantly reduce the expertise barrier for framework deployment and improve its scalability in real-world applications.

The advancement of large-scale distributed testing technology will enable support for more complex application scenarios. By integrating federated learning and edge computing technologies, researchers can achieve collaborative testing across institutions and geographical boundaries while preserving data privacy and security. This approach addresses the growing need for comprehensive testing in distributed AI systems and enables broader participation in quality assurance efforts without compromising sensitive data protection requirements.

## References


[1] J. Wang, Y. Huang, C. Chen, Z. Liu, S. Wang, and Q. Wang, "Software testing with large language models: Survey, landscape, and vision," *IEEE Transactions on Software Engineering*, 2024.

[2] K. Li and Y. Yuan, "Large language models as test case generators: Performance evaluation and enhancement," *arXiv preprint arXiv:2404.13340*, 2024.

[3] W. Wang, C. Yang, Z. Wang, Y. Huang, Z. Chu, D. Song, and L. Ma, "Testeval: Benchmarking large language models for test case generation," *arXiv preprint arXiv:2406.04531*, 2024.

[4] H. F. Chang and M. Shokrolah Shirazi, "A systematic approach for assessing large language models' test case generation capability," *Software*, vol. 4, no. 1, p. 5, 2025.

[5] K. Liu, Y. Liu, Z. Chen, J. M. Zhang, Y. Han, Y. Ma, and G. Huang, "Llm-powered test case generation for detecting tricky bugs," *arXiv preprint arXiv:2404.10304*, 2024.

[6] S. Rehan, B. Al-Bander, and A. Al-Said Ahmad, "Harnessing large language models for automated software testing: A leap towards scalable test case generation," *Electronics*, vol. 14, no. 7, 2025.

[7] M. Ferreira, L. Viegas, J. P. Faria, and B. Lima, "Acceptance test generation with large language models: An industrial case study," *arXiv preprint arXiv:2504.07244*, 2025.

[8] S. K. S. Hari, V. V. R. Konda, V. Kamakoti, V. M. Vedula, and K. S. Maneperambil, "Automatic constraint based test generation for behavioral HDL models," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 4, pp. 408–421, 2008.

[9] J. Liu, R. Liang, X. Zhu, Y. Zhang, Y. Liu, and Q. Liu, "LLM4TDG: test-driven generation of large language models based on enhanced constraint reasoning," *Cybersecurity*, vol. 8, no. 1, pp. 1–23, 2025.

[10] C. Tessler, D. J. Mankowitz, and S. Mannor, "Reward constrained policy optimization," *arXiv preprint arXiv:1805.11074*, 2018.

[11] J. Kim, M. Kwon, and S. Yoo, "Generating test input with deep reinforcement learning," in *IEEE/ACM 11th International Workshop on Search-Based Software Testing*, 2018.

[12] L. Peng, P. Song, X. Jiang, and L. Xie, "Model-based interlocking software test case generation method," *Railway Signalling & Communication Engineering*, vol. 19, no. 11, 2022.

[13] J. Song, X. Zuo, X. Zhang, and H. Huang, "Comprehensive survey of large language model evaluation methods," *Aerospace Measurement Technology*, vol. 45, no. 2, p. 1, 2025.

[14] C. Qin, S. Chen, K. Li, H. Liu, L. Yang, and Z. Ma, "Automatic generation technology of white-box unit test cases for MC/DC coverage," *Science Technology & Engineering*, vol. 24, no. 30, 2024.

[15] B. Steenhoek, M. Tufano, N. Sundaresan, and A. Svyatkovskiy, "Reinforcement learning from automatic feedback for high-quality unit test generation," *arXiv preprint arXiv:2310.02368*, 2023.

[16] B. Yang, J. Wu, L. Xu, K. Bi, and C. Liu, "A software testing requirement modeling and test case generation method," *Journal of Computer Research and Development*, vol. 37, no. 3, pp. 522–538, 2014.

[17] C. Tan, R. Behjati, and E. Arisholm, "Enhancing synthetic test data generation with language models using a more expressive domain-specific language," in *ICTSS 2023 Proceedings*, pp. 15–30, 2023.

[18] Z. Qian, Q. Yu, D. Zhang, C. Yao, L. Qin, and W. Cheng, "Chain-type multi-path coverage test case generation combining SVM and XGBoost," *Journal of Software*, vol. 35, no. 6, pp. 2795–2820, 2023.

[19] S. Kang, J. Yoon, and S. Yoo, "Large language models are few-shot testers: Exploring llm-based general bug reproduction," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, IEEE, 2023, pp. 2312–2323.

[20] N. Hansen, H. Su, and X. Wang, "TD-MPC2: Scalable, robust world models for continuous control," *arXiv preprint arXiv:2310.16828*, 2023.