Information and Software Technology 189 (2026) 107944

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

ISVMT: An approach of indicator systems validation based on metamorphic testing and data mutation[☆]

GuoHao Ma[a], Bo Yang[a,*], XiaoKai Xia[b], Luo Xu[b,*]

[a] School of Information Science and Technology, Beijing Forestry University, Beijing 100083, China
[b] China Electronics Technology Group Corporation Information Science Research Institute, Beijing 100042, China

ARTICLE INFO

Dataset link: https://pan.baidu.com/s/1rjgDKreFjLyA9FckObgmyQ?pwd=32bi

Keywords:
Software testing
Indicator systems
MT
Software validation
Data mutation

ABSTRACT

Context: Indicator systems are pivotal in assessing software quality, conducting economic analysis, and various other domains. However, their validation is frequently hindered by the absence of explicit expected outputs, posing a significant challenge to their reliability and effectiveness.
Objective: This paper aims to address this challenge by proposing a novel validation method for indicator systems based on metamorphic testing (MT). The method seeks to eliminate the oracle problemby generating follow-up test cases and verifying their consistency through designed logical constraints, known as metamorphic relations, between inputs and outputs.
Methods: To validate the proposed method, we conducted empirical research on three diverse cases: software quality evaluation, red wine quality assessment, and ecological and economic benefit evaluation. We designed metamorphic relations specific to each case and applied our method to identify inconsistencies and potential errors in the indicator systems.
Results: Our experiments successfully identified errors in real-world indicator systems, demonstrating the method's capability to detect flaws that traditional methods might overlook. Furthermore, we performed mutation testing, achieving an average mutation score of 0.83 on the mutation dataset, which significantly outperforms the traditional statistical analysis method with a maximum score of 0.65.
Conclusion: This paper presents a generalized solution for the validation of indicator systems lacking clear expectations, offering substantial application value in the fields of software engineering and decision support. The proposed method not only increases confidence in the reliability of indicator systems but also opens up promising research avenues for further improving the accuracy and efficiency of validation processes in complex domains.

1. Introduction

Indicator systems [1,2], or indicator frameworks, are structured sets of measurable factors or metrics that are used to assess, monitor, and evaluate various aspects of a system, process, or phenomenon. These systems are designed to provide a quantitative and objective means of analysis, which facilitates informed decision-making. Indicator systems have become indispensable across a spectrum of fields, extending from the realm of software quality assessment [3,4], artificial intelligence evaluation [5,6], software reliability analysis [7] and natural disaster risk assessment [8]. They also play a pivotal role in gauging market trends [9] and economic health [10], thereby informing policy effectiveness. Additionally, they monitor environmental conditions [11] to ensure ecological health and sustainability, assess healthcare system performance and patient outcomes [12], and measure the impact of teaching methods on student achievements [13].

However, validating indicator systems presents significant challenges due to the absence of explicit expected outputs. The validation process is a challenge recognized as the Test Oracle problem [14]. This lack of clear benchmarks makes it difficult to assess the reliability and effectiveness of these systems. Traditional validation methods often rely on manual labeling or predefined expected results [15,16], which may not always be available or feasible. This gap necessitates innovative approaches that can validate indicator systems without requiring explicit expected outputs.

Metamorphic testing (MT) [17] has emerged as a robust strategy to mitigate this issue. Xie et al. [18] successfully applied MT to address the

☆ This article is part of a Special issue entitled: 'Special Issue on MT' published in Information and Software Technology.
* Corresponding authors.
E-mail addresses: maguohao@bjfu.edu.cn (G. Ma), yangbo@bjfu.edu.cn (B. Yang).

https://doi.org/10.1016/j.infsof.2025.107944
Received 20 March 2025; Received in revised form 27 July 2025; Accepted 20 October 2025
Available online 24 October 2025
0950-5849/© 2025 Published by Elsevier B.V.

Test Oracle problem in machine learning verification, proposing a suite of Metamorphic Relations (MRs) [19] and conducting experimental studies within a widely-used machine learning framework, effectively identifying system errors. Furthermore, to tackle the verification and interpretation of autonomous vehicle (AV) decision-making in the absence of ground-truth labels, Deng et al. [20] introduced the Sequential MT (SMART) framework. This approach, grounded in MT principles, systematically assesses AV decision logic by generating synthetic test case sequences across various driving scenarios and weather conditions.

MT for machine learning (ML) models typically involves model-specific MRs tailored to the architecture or learning objective, such as input perturbation or label flipping. In contrast, MT for indicator systems uses domain-invariant MRs derived from logical properties like monotonicity or unit invariance, allowing validation without reverse-engineering the system.

In light of the multifaceted challenges associated with validating indicator systems, we introduce ISVMT, an innovative approach to validation that is firmly grounded in the principles of MT. This method is designed to navigate the complexities of indicator systems by providing a structured strategy for assessing their accuracy and reliability. Our approach involves an in-depth exploration of how MT can be specifically applied to validate indicator systems, and we rigorously evaluate its effectiveness in this context. At the core of our method is the proposition of a set of tailored metamorphic relations that are crafted to align with the unique characteristics of the indicator system that is subject to validation. These relations serve as a blueprint, outlining the expected alterations in outputs that should correspond to modifications in inputs.

The process begins with the employment of conventional test case generation techniques to establish a baseline of source test cases. These initial cases serve as the foundation upon which subsequent follow-up test cases are developed. Guided by the predefined metamorphic relations, these follow-up test cases are designed to probe the system's response to controlled input variations. The crux of our validation strategy lies in the comparative analysis of the output relationships between the source and follow-up test cases. This comparison is pivotal for determining system integrity; conformity of these outputs to the anticipated metamorphic relations affirms system correctness, while any deviation acts as a red flag, indicating a potential fault within the system.

The main contributions of this paper are three-fold:

- Pioneering Feasibility Exploration: We are the first to investigate the feasibility of MT for the validation of indicator systems, meticulously examining how to effectively implement this innovative testing approach in the context of indicator system validation.
- Tailored Metamorphic Relation Design: We have developed specific metamorphic relations that are applicable to indicator systems, ensuring that the testing methodology is both relevant and effective for this particular domain.
- Comprehensive Case Study and Comparative Analysis: we undertook extensive case studies leveraging real-world indicator systems to evaluate the efficacy of our MT approach. Beyond traditional validation methods, we also conducted a comparative analysis with validation techniques grounded in large language models. Our empirical findings robustly illustrate the superior effectiveness of our MT approach in the validation of indicator systems.

The remainder of this paper is organized as follows. Section 2 briefly introduces the background of MT and mutation testing. Section 3 presents our proposed ISVMT. Section 4 reports the empirical study for the evaluation of ISVMT and Section 5 describes the experiment's result. We provide discussions and lessons learned in Section 6. Section 7 briefly introduces the related works. Section 8 summarizes the paper.

2. Background

2.1. MT

To address the Test Oracle problem, Chen et al. [21] proposed the concept of MT. MT recognizes that test cases that do not reveal faults (i.e., successful test cases) also contain useful information, which can be leveraged to generate new test cases for more thorough program verification. MT evaluates a program by examining the logical relationships between the execution results of successful test cases and those of newly generated test cases derived from them. These logical relationships, known as metamorphic relations (Definition 1), eliminate the need to construct expected outputs. The metamorphic relations that should hold between program execution results are functional properties of the program and can be derived from its specification. If a program P is designed to compute a function f, then f naturally serves as the specification that P must adhere to, and the relations for verifying the correctness of the program can be derived from f.

Definition 1 (*Metamorphic Relations (MRs)*)**.** Suppose a program P is designed to compute a function f, where $x_1, x_2, \dots, x_n$ $(n > 1)$ are n sets of inputs to f, and $f(x_1), f(x_2), \dots, f(x_n)$ are their corresponding function outputs. If the inputs $x_1, x_2, \dots, x_n$ satisfy a relationship r, then their corresponding outputs $f(x_1), f(x_2), \dots, f(x_n)$ should satisfy a corresponding relationship r_f, that is:

$$r(x_1, x_2, \dots, x_n) \implies r_f(f(x_1), f(x_2), \dots, f(x_n)) \tag{1}$$

Then, (r, r_f) is called a metamorphic relation of P.

Obviously, if P is correct, it must satisfy the following derivation:

$$r(I_1, I_2, \dots, I_n) \implies r_f(P(I_1), P(I_2), \dots, P(I_n)) \tag{2}$$

where $I_1, I_2, \dots, I_n$ are the inputs to program P corresponding to $x_1, x_2, \dots, x_n$, and $P(I_1), P(I_2), \dots, P(I_n)$ are the corresponding outputs. Therefore, the correctness of program P can be determined by verifying whether Eq. (2) holds. MT is a testing approach based on metamorphic relations.

Suppose that when testing a program P that computes the sine function, an input $t = 57.3°$ is used, and the corresponding output is $P(t) = 0.8415$. Since the exact value of $\sin(57.3°)$ is difficult to determine, it is not easy to directly verify the correctness of $P(t)$. Using MT to check the correctness of program P, we can construct a metamorphic relation based on the mathematical property $\sin(x) = \sin(\pi - x)$. According to this property, a follow-up test case (Definition 2) for t is given by $t' = 180° - 57.3° = 122.7°$. Executing P with t' as input yields an output of $P(t') = 0.8402$. By comparing $P(t)$ and $P(t')$, it is evident that they do not satisfy the expected metamorphic relation, indicating the presence of an error in program P.

Definition 2 (*Source/Follow-up Test Cases*)**.** When using the metamorphic relation (r, r_f) to test program P, the initially given test case (generated by other test case generation methods) is referred to as the *source test case*. A test case that is calculated based on the source test case according to the relation r is called a *follow-up test case* with respect to the metamorphic relation (r, r_f).

2.2. Mutation testing

Mutation testing [22] is a fault-based software testing technique that involves introducing small changes, known as mutations, into a program's source code to create modified versions called mutants. These mutants are then tested to evaluate the effectiveness of existing test cases in detecting such faults. The primary objective of mutation testing is to assess and enhance the quality of test suites by determining their ability to identify injected faults. A robust test suite should detect discrepancies between the original program and its mutants, thereby "killing" the mutants. If a mutant remains undetected, it indicates potential inadequacies in the test suite, necessitating the development of additional test cases to improve fault detection capabilities.

Program mutation is performed under the guidance of mutation operators, which define syntactic modification rules for introducing changes to programs. For instance, operator mutation involves arithmetic operator replacement (e.g., substituting '+' with '-'). Mutation testing generates a set of mutants based on a selected collection of mutation operators for subsequent execution analysis. The testing objective aims to kill all generated mutants. In mutation testing, the ratio of non-equivalent mutants killed by the test case suite to the total number of non-equivalent mutants is defined as the Mutation Score (MS), a critical metric for evaluating the defect detection capability of the test case suite. In this study, a mutant is considered killed if it violates any metamorphic relation. The calculation method for the mutation score is provided in Formula (3).

$$MS(MUT, TS) = \frac{|MUT_{killed}|}{|MUT| - |MUT_{eq}|} \tag{3}$$

Where TS represent the test suite, $|MUT_{\text{killed}}|$ represent the number of mutants killed by TS, $|MUT|$ represent the total number of mutants, and $|MUT_{\text{eq}}|$ represent the number of equivalent mutants. The value of $MS(MUT, TS)$ lies between 0 and 1. A higher value indicates that more mutants have been killed by the test suite, and the defect detection capability of the test suite is stronger. Conversely, a lower value indicates weaker defect detection capability. When $MS(MUT, TS) = 0$, it means that the test suite has not killed any mutants. When $MS(MUT, TS) = 1$, it means that the test suite has killed all non-equivalent mutants.

3. Methodology

3.1. The framework of ISVMT

Fig. 1 illustrates the overall workflow of ISVMT. ISVMT consists of two phases:

In the first phase, we employ metamorphic testing to detect potential faults in the indicator system. The specific process involves the following four steps: (1) properties analysis; (2) MRs identification; (3) test case generation; and (4) MR verification. During this process, the violation of any MR indicates the discovery of a potential issue. If no issues are detected in this phase, there may be three possible reasons: (1) the indicator system itself is indeed free of defects; (2) the proposed metamorphic relations fail to cover all the functionalities under test; or (3) the test cases lack sufficient coverage to effectively trigger latent errors.

To enhance users' confidence in the verification results, we designed a second phase—mutation analysis—to assess the adequacy of the verification performed in the first phase. This phase is optional. If issues are identified in the first phase, it suggests that the indicator system may contain genuine defects, making further manual error injection unnecessary.

However, if no faults are detected in the first phase, the second phase proceeds by injecting artificial errors into the original indicator system to generate mutants. The test cases generated in the first phase are then used to test these mutants. The mutation score is calculated to quantify the fault detection capability of the first phase, thereby evaluating the effectiveness of metamorphic testing. Based on the mutation score, a decision is made on whether to supplement new metamorphic relations or expand the test case suite.

3.2. Validation of the original indicator system

Definition 3 (*Mutation Operator (MO)* [23]). Mutation Operator can be defined as a function MO, which is capable of altering the input data of test cases. The output generated by the mutation operator is denoted as MOUT.

$$MOUT = MO(TD, FLD)$$

that syntactically alters the chosen field FLD of the test data TD while preserving schema validity.

Definition 4 (*Mutation Rule (MuR)*). A quadruple

$$MuR = \langle TD, FLD, MO, MOR \rangle$$

where MOR supplies the numerical parameter(s) for MO.

In Phase 1, we assume the indicator system may be faulty and apply metamorphic testing to expose latent defects. The phase comprises four tightly-coupled steps:properties analysis, MRs identification; test case generation; and MR verification.

The first step is Inherent-Property Analysis. An inherent property is a domain-independent logical requirement that the indicator system must satisfy regardless of its concrete implementation. For example, in a weighted-scoring model the following must always hold: If the weight of an indicator is zero, any change in the value of that indicator shall not affect the final score. These properties serve as functional points to be tested and supply the domain knowledge required to identify MRs.

Next step is MRs Identification. From each inherent property we identify an MR that formally prescribes the expected output change when an input transformation is applied. We adopt the standard notation

MR: $r(x_1, \dots, x_k) \Rightarrow r_f(f(x_1), \dots, f(x_k))$

where r is the input constraint and r_f is the expected output relation. We summarized 9 generic MRs applicable to any indicator system; they are independent of the underlying algorithm.

The third step is Test Case Generation. Source test cases T_s are generated by classical techniques such as equivalence partitioning, boundary-value analysis, and random test input generation. Follow-up test cases are generated by data mutation(DM) guided by the selected MR and mutation operator (Definition 3). For example, CR relation assume that if an indicator is a positive indicator (i.e., larger values are better), then increasing a sample's value on this indicator should lead to a higher evaluation result for that sample, while the evaluation results of other samples should remain unchanged. Let $T_s = (x_1, x_2, x_3)$ be a source test case with positive indicators. Choosing FLD=x_1(maintainability), MO=MO-DoubleAdd [23], and MOR = 0.1 .yields $T_f = (x_1{+}0.1, x_2, x_3)$. With m positive indicators and n admissible increments, MO produces $m \times n$ distinct follow-up test case.

Finally, we apply the verification and get the report. In this step, we execute the indicator system on (T_s, T_f) and compare outputs (y_S, y_f,). Any violation of the prescribed MR constitutes a detected anomaly and is logged in the test report.

3.3. Indicator system mutation analysis

Phase 2 augments Phase 1 by deliberately injecting faults—via program mutation—to assess the detection capability of the MRs and test cases. If Phase 1 already reveals a violation, Phase 2 is skipped.

Firstly, we need to mutate the Indicator System. Mutants are generated by applying mutation operators MO_{prog} to the indicator system's source code or computation logic:

$$ISm = MUT(IS, MOprog)$$

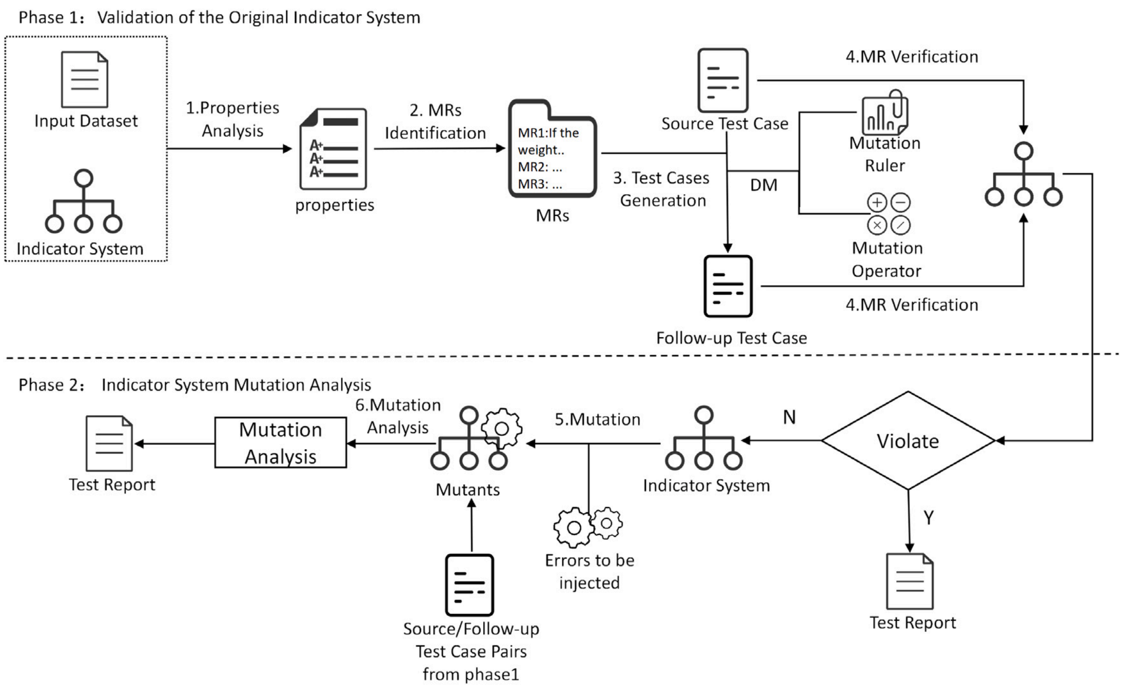

Fig. 1. Workflow of ISVMT.

where IS is the original indicator system and MO_{prog} may perform operator replacement, parameter alteration, or formula tampering. Equivalent mutants (those semantically identical to IS) are filtered out, leaving only non-equivalent mutants for evaluation.

Secondly, we conduct the mutation analysis. Reuse the MRs and test cases from Phase 1 to execute each mutant. A mutant is killed if at least one MR is violated. The mutation score(MS) quantifies the effectiveness of the test suite. A high mutation score ($MS \approx 1$) indicates strong detection power and increases confidence in Phase 1's conclusion that the system itself is likely correct. A low score signals insufficient MR coverage or inadequate test-case diversity, prompting refinement of both.

3.4. Metamorphic relations for indicator systems

We proposed MRs for indicator systems based on both domain knowledge and users' general expectations of indicator system behavior, rather than specific algorithmic implementations. MRs can be identified from two sources: (1) inherent properties that should be satisfied by the algorithmic implementation of the indicator system, and (2) general behavioral expectations of evaluation tools held by users, developers, or domain experts. The former type of MRs is primarily used for verification, while the latter, although not necessarily required by all implementations, can be used for validation [18]—that is, to assess whether the current implementation meets reasonable user expectations or potential needs.

When applying the metamorphic relations proposed in this section to a specific indicator system, appropriate adjustments should be made based on the characteristics of the system's input. Research has shown that overly generic metamorphic relations may be ineffective for testing [24]. Therefore, researchers and practitioners are encouraged to supplement, extend, or adapt these relations according to the specific features and business logic of their indicator systems, in order to ensure better alignment with the application context.

Notation:

We abstract the indicator system as a function $y = f(X, w)$.

Let $X \in \mathbb{R}^{m\times n}$ ($n \geq 1,\ m \geq 1$) denote the decision matrix, where each row represents an evaluated object (sample), and each column corresponds to an indicator (i.e., an attribute of the object); let $w \in \mathbb{R}^n$ be the weight vector for the n indicators and $y \in \mathbb{R}^m$ represents the evaluation results of the m samples.

Source Input: $Input_s = (X_s, w_s)$

Follow-up Input: $Input_f = (X_f, w_f)$

Source Output: $y_s = f(X_s, w_s) \in \mathbb{R}^m$

Follow-up Output: $y_f = f(X_f, w_f) \in \mathbb{R}^m$

1. Zero-weight Invariance Relation (ZIR). If the weight of a certain indicator is zero, then modifying the input value of this indicator should not affect the output. Therefore, the follow-up output should be equal to the source output.

Principle explanation: In the design of an indicator system, the information carried by each indicator may contribute differently to the overall evaluation objective. Weights are used to reflect the relative importance of each indicator in the overall assessment, thereby capturing such differences. Assigning a weight of zero to a particular indicator indicates that this indicator is currently considered irrelevant or intentionally excluded from the evaluation. Therefore, the system must ensure that such "masked indicators" have no impact on the final evaluation result, accurately reflecting the intended role of weights. When the weight of an indicator is zero, its value should be completely ignored by the evaluation function. As a result, any modification to the value of this indicator should not affect the evaluation outcome.

Source Input: $Input_s = (X_s, w_s)$, satisfying:

- There exists an indicator $j \in \{1, 2, \dots, n\}$ with $w_s[j] = 0$.

Follow-up Input: $Input_f = (X_f, w_f)$, satisfying the following conditions:

- For all $i \in \{1, 2, \dots, m\}$, $X_f[i][j] \neq X_s[i][j]$.
- For all $i \in \{1, 2, \dots, m\}$ and all $k \neq j$, $X_f[i][k] = X_s[i][k]$.
- $w_f = w_s$.

Expected result: $y_f = y_s$.

2. Scale Invariance Relation (SIR). The indicator system should eliminate the influence of measurement units. If the unit of an indicator is changed (e.g., converting 1 km to 1000 m), which is equivalent to multiplying the indicator values by a positive constant ($\alpha > 0$), the results should remain unchanged.

Principle explanation: The underlying assumption of this metamorphic relation is that the indicator system depends not on the absolute values of the indicators, but rather on the relative relationships among objects for each indicator—such as rankings, differences, or relative magnitudes. In the indicator system, indicators often have different scales and ranges of values, and direct weighting calculations can easily lead to a disproportionate impact on the results of indicators with larger scales, thus obscuring the importance of other indicators. The indicator system should eliminate gaps in the scales so as to realize the comparability of indicators of different scales and levels. When units are converted, the raw data are rescaled accordingly. If such unit conversions lead to changes in the evaluation result, it indicates a lack of robustness or scale-invariance in the indicator system. Therefore, this metamorphic relation is also commonly used to test the system's robustness to unit normalization.

Transformations such as normalization and standardization are designed to eliminate the influence of scale. Consequently, if all values of indicator j are multiplied by a non-zero constant k, the relative relationships — such as ranking or variance structure — remain unchanged, and the resulting evaluation outcome should also remain unaffected.

Source Input: $Input_s = (X_s, w_s)$

Follow-up Input: $Input_f = (X_f, w_f)$, satisfying the following conditions:

- There exists an indicator $j \in \{1, 2, \dots, n\}$ and a constant $k \neq 0$, such that for all $i \in \{1, 2, \dots, m\}$,
  - $X_f[i][j] = k \cdot X_s[i][j]$
  - $X_f[i][l] = X_s[i][l]$ for all $l \neq j$
- The weight vector remains unchanged: $w_f = w_s$

Expected result: $y_f = y_s$

3. Correlation Relation (CR). If an indicator is a positive indicator (i.e., larger values are better), then increasing a sample's value on this indicator should lead to a higher evaluation result for that sample, while the evaluation results of other samples should remain unchanged.

Principle explanation: A positive indicator refers to an attribute where a higher indicator value implies better performance of the sample. For example, in scenarios such as "output value", "income", or "accuracy", higher values are preferable. Therefore, when a sample achieves a higher score on a positive indicator, it is expected to receive a higher or at least not lower overall evaluation result.

Source Input: $Input_s = (X_s, w_s)$, satisfying:

- The jth indicator is a positive-oriented indicator with weight $w_s[j] > 0$

Follow-up Input: $Input_f = (X_f, w_f)$, satisfying the following conditions:

- Only modify the value of the jth indicator for the pth sample, such that: $X_f[p][j] > X_s[p][j]$,
- Keeping all other positions unchanged: $X_f[i][l] = X_s[i][l]$ for all $i \neq p$ or $l \neq j$
- $w_f = w_s$

Expected Result: $y_s[p] \leq y_f[p]$ and $y_s[i] = y_f[i]$ for all $i \neq p$

4. Remove Sample (RS). If a sample is removed without affecting the attribute information of the remaining samples, the evaluation results of the remaining samples should remain unchanged.

Principle explanation: If users expect the indicator system to evaluate each sample independently—meaning that the evaluation result y_i for each sample depends only on that sample's own indicator values and weights w_i, and not on the presence or attributes of other samples—then the indicator system should not include mechanisms that rely on referencing other samples' values (such as rank normalization or quantile scoring). Otherwise, this metamorphic relation would not hold. However, if the evaluation function uses global information from all samples during computation (e.g., maximum or average values), removing a sample may cause changes in the overall evaluation results. In this case, the MR can be used to detect whether the indicator system depends on the entire sample set.

Source Input: $Input_s = (X_s, w_s)$

Follow-up Input: $Input_f = (X_f, w_f)$, satisfying the following conditions:

- There exists a sample index $p \in \{1, 2, \dots, m\}$ such that the sample $X_s[p]$ is removed, resulting in:

  $$X_f \in \mathbb{R}^{(m-1)\times n}, \quad X_f = X_s \setminus \{X_s[p]\}$$
- $w_f = w_s$

Expected result: For all $i \in \{1, 2, \dots, m\}$,

$$y_f[i'] = y_s[i], \quad \text{where } i' = \begin{cases} i, & i < p \\ i-1, & i > p \end{cases}$$

5. Maximum Value Insertion Relation (MAXR). If a new sample is added, and for each indicator, its value is set to the best among all existing samples (i.e., the maximum for positive indicators and the minimum for negative indicators), then its evaluation result should be greater than those of all existing samples.

Principle explanation: If an object outperforms all others on every indicator, its evaluation result should be higher. This represents the most fundamental guarantee of fairness and validity in an evaluation system.

Source Input: $Input_s = (X_s, w_s)$

Follow-up Input: $Input_f = (X_f, w_f)$, satisfying the following conditions:

- $X_f \in \mathbb{R}^{(m+1)\times n}$, and for all $j \in \{1, 2, \dots, n\}$,

  $$\begin{cases} X_f[m+1][j] \geq \max_{1\leq i\leq m} X_s[i][j], & \text{if } j \text{ is a positive indicator} \\ X_f[m+1][j] \leq \min_{1\leq i\leq m} X_s[i][j], & \text{if } j \text{ is a negative indicator} \end{cases}$$
- $w_f = w_s$

Expected result: $y_f[m+1] \geq y_s[i], \ \forall i \in \{1, 2, \dots, m\}$

6. Minimum Value Insertion Relation (MINR). If a new sample is added, and for each indicator, its value is set to the worst among all existing samples (i.e., the minimum for positive indicators and the maximum for negative indicators), then its evaluation result should be lower than those of all existing samples

Principle explanation: If an object scores lower than others on all indicators, its evaluation result should naturally be the lowest. This ensures the most fundamental fairness and validity of the evaluation system.

Source Input: $Input_s = (X_s, w_s)$

Follow-up Input: $Input_f = (X_f, w_f)$, satisfying the following conditions:

- $X_f \in \mathbb{R}^{(m+1)\times n}$, and for all $j \in \{1, 2, \dots, n\}$,

  $$\begin{cases} X_f[m+1][j] \geq \max_{1\leq i\leq m} X_s[i][j], & \text{if } j \text{ is a negative indicator} \\ X_f[m+1][j] \leq \min_{1\leq i\leq m} X_s[i][j], & \text{if } j \text{ is a positive indicator} \end{cases}$$
- $w_f = w_s$

Expected result: $y_f[m+1] \leq y_s[i], \ \forall i \in \{1, 2, \dots, m\}$

7. Indicator Swapping Invariance Relation (ISIR). If two indicators are swapped and their respective weights are swapped accordingly, while all other parts remain unchanged, the evaluation results should remain the same.

Principle explanation: As long as each indicator's value is correctly paired with its corresponding weight, the order of the indicators should not influence the final outcome.

Source Input: $Input_s = (X_s, w_s)$

Follow-up Input: $Input_f = (X_f, w_f)$, satisfying the following conditions:

- Select any two indicators $j_1, j_2 \in \{1, 2, \dots, n\}$, where $j_1 \neq j_2$.
- For all $i \in \{1, 2, \dots, m\}$, $X_f[i][j_1] = X_s[i][j_2]$, $X_f[i][j_2] = X_s[i][j_1]$
- $w_f[j_1] = w_s[j_2]$, $w_f[j_2] = w_s[j_1]$
- All other elements remain unchanged.

Expected result: $y_f = y_s$

8. Reordering Invariance Relation (RIR). If the order of two samples (i.e., two rows in the data matrix) is swapped, the output of the evaluation function should be swapped accordingly, but the result value corresponding to each sample should remain unchanged.

Principle explanation: In an indicator system, the score of each sample depends solely on its own indicator values and the corresponding weights, rather than the sample's position in the input. Therefore, swapping the order of two samples does not affect their individual evaluation results — only the order of the output changes accordingly.

Source Input: $Input_s = (X_s, w_s)$

Follow-up Input: $Input_f = (X_f, w_f)$, satisfying:

- Select any two evaluation objects $i_1, i_2 \in \{1, 2, \dots, m\}$, $i_1 \neq i_2$
- for all $j \in \{1, 2, \dots, n\}, X_f[i_1][j] = X_s[i_2][j], \quad X_f[i_2][j] = X_s[i_1][j]$
- for all $i \notin \{i_1, i_2\}$, $X_f[i][j] = X_s[i][j]$
- $w_f = w_s$

Expected Output: $y_s[i_1] = y_f[i_2], y_s[i_2] = y_f[i_1]$

9. Weight Increase Relation (WIR). When the weight of a positive indicator is increased while the weights of all other indicators remain unchanged: The overall evaluation scores of all samples should increase accordingly. Samples with higher values on this indicator should exhibit a greater increase in their evaluation results.

Principle explanation: In an indicator system, although the final evaluation score may involve further processing beyond a simple weighted sum (such as normalization, aggregation, or transformation), the fundamental role of weights is to adjust the contribution of each indicator by multiplying them with the corresponding indicator values. For a positive indicator (i.e., higher values indicate better performance): Increasing the weight of this indicator effectively amplifies its influence in the evaluation process. Among all samples, those with higher values for this indicator will benefit more from this amplification, leading to higher evaluation results. Since the weights of other indicators remain unchanged, the change in evaluation is mainly driven by the values of the adjusted indicator. As a result, the overall evaluation scores of all samples should increase, with samples that have higher values on this indicator experiencing a greater increase.

Source input: $Input_s = (X_s, w_s)$

Follow-up input: $Input_f = (X_f, w_f)$, such that:

- $X_f = X_s$
- There exists a positive indicator $j \in \{1, 2, \dots, n\}$ such that $w_f[j] > w_s[j]$, and for all $l \neq j$, $w_f[l] = w_s[l]$

Expected result: For any two samples: $i_1, i_2 \in \{1, 2, \dots, m\}$ if $X_s[i_1][j] > X_s[i_2][j] \Rightarrow y_f[i_1] - y_s[i_1] > y_f[i_2] - y_s[i_2]$

4. Experiments

In order to demonstrate the validity of our approach in the validation of indicator systems, we selected indicator systems from different domains for case studies.

4.1. Research question

- **RQ1**: How effective is ISVMT on the original datasets?
- **RQ2**: How effective is ISVMT on mutated datasets?
- **RQ3**: How does the verification effectiveness of ISVMT compared to other methods?
- **RQ4**: How does the verification effectiveness of ISVMT compare to LLMs?

4.2. Dataset

We selected indicator systems from three related studies for our case study, Table 1 presents the three indicator systems along with their abbreviations, code scale, and a brief description.

Software Quality Evaluation Indicator System (SQ). The SQ [25] adopts the Fuzzy Comprehensive Evaluation (FCE) method and the Triangular Fuzzy Number Analytic Hierarchy Process (TFN-AHP). The TFN-AHP is employed to determine the weights of each layer or group of indicators, followed by the application of FCE to assess software quality. Table 2 presents the specific indicators and their corresponding weights.

Wine Quality Evaluation Indicator System (Wine). Cortez et al. [26] developed a wine quality assessment model that quantifies the relationship between the physicochemical properties of wine and human sensory preferences (i.e., taste evaluation) through data mining techniques, enabling an objective prediction of wine quality. Specifically, the model constructs a predictive framework using regression analysis methods (e.g., support vector machines, neural networks, etc.) based on a large dataset [1] of physicochemical test results (such as alcohol content, acidity, and sugar levels) and wine tasters' ratings.

Ecological-Economic Efficiency Evaluation Indicator System (Eco). To evaluate the impact of green technology innovation on the eco-economic efficiency of strategic emerging industries, an evaluation index system was established using the entropy-weight TOPSIS method [27]. Table 3 presents the specific indicators. By analyzing the factors influencing the innovation of green technology, the system incorporates both positive and hindering factors. The proposed evaluation process reduces subjectivity in weight assignment, addresses limitations related to sample size and data distribution, and helps researchers accurately identify shortcomings based on the evaluation results.

4.3. Metamorphic relations selection

In Section 3.4, we introduced generic metamorphic relations applicable to the indicator system that are directly usable as a repository of metamorphic relations. For Wine and Eco, we used all nine metamorphic relations; for SQ,all were used except SIR, RS, MAXR, and MINR, because these MRs were not applicable to this particular indicator system. Specifically, the input of this system consists of manual scores assigned to each indicator, with identical value ranges and no differences in units or scales, making SIR (Scale Invariance Relation) inapplicable. Moreover, this indicator system evaluates only one sample at a time, which prevents the application of RS, MAXR, and MINR relations.

[1] https://www.datacastle.cn/dataset_description.html?id=756&type=dataset

Table 1
Experimental datasets.

Dataset	LOC	Dataset description
Software Quality Evaluation Indicator System (SQ)	66	An indicator system for evaluating software quality.
Wine Quality Evaluation Indicator System (Wine)	166	An indicator system for evaluating wine quality.
Ecological-Economic Efficiency Evaluation Indicator System (Eco)	228	An indicator system for evaluating industrial eco-economic benefits using the Topsis method.

Table 2
ISO/IEC 25010 software quality evaluation model and weights.

Characteristic	Weight	Sub-characteristic	Weight
Functional Suitability (B_1)	0.2029	Functional Completeness (C_{11})	0.5394
		Functional Appropriateness (C_{12})	0.2835
		Functional Correctness (C_{13})	0.1771
Performance Efficiency (B_2)	0.2029	Time Behavior (C_{21})	0.4568
		Resource Utilization (C_{22})	0.2963
		Capacity (C_{23})	0.2469
Compatibility (B_3)	0.1045	Interoperability (C_{31})	0.6667
		Coexistence (C_{32})	0.3333
Usability (B_4)	0.1487	Recognizability (C_{41})	0.3751
		Learnability (C_{42})	0.2436
		Operability (C_{43})	0.1505
		User Error Prevention (C_{44})	0.1031
		Aesthetic Integration (C_{45})	0.0056
		Accessibility (C_{46})	0.1221
Reliability (B_5)	0.2029	Maturity (C_{51})	0.2316
		Fault Tolerance (C_{52})	0.4392
		Recoverability (C_{53})	0.2109
		Security (C_{54})	0.1183
Security (B_6)	0.0705	Confidentiality (C_{61})	0.1185
		Integrity (C_{62})	0.4655
		Traceability (C_{63})	0.2597
		Responsibility (C_{64})	0.0684
		Authenticity (C_{65})	0.088
Maintainability (B_7)	0.0426	Modularity (C_{71})	0.1285
		Reusability (C_{72})	0.4655
		Analyzability (C_{73})	0.2497
		Modifiability (C_{74})	0.0782
		Testability (C_{75})	0.0782
Portability (B_8)	0.025	Adaptability (C_{81})	0.5688
		Installability (C_{82})	0.2303
		Replaceability (C_{83})	0.2009

Table 3
The evaluation indicators system of ecological-economic efficiency.

Target layer	First-level indicators	Secondary-level indicators
Ecological-Economic Efficiency	Innovation policy I_1	Organization learning I_{11}, Enterprise organization form I_{12}, Enterprise inertia I_{13}, The degree of path dependence I_{14}
	Eco-environment efficiency I_2	Environmental pollution treatment intensity I_{21}, Waste water, waste gas and solid waste emissions I_{22}, Green environmental protection benefits I_{23}
	Utilization of energy and resources I_3	Total consumption of energy and resources I_{31}, Occupancy rate of ecological resources I_{32}, New energy absorption capacity I_{33}
	Green technology innovation production I_4	Green sales rate I_{41}, Energy consumption reduction rate I_{42}, The proportion of green technology I_{43}
	Eco-social efficiency I_5	The market share of green products I_{51}, “Three wastes” treatment effect I_{52}, Population carrying capacity of environment I_{53}

4.4. Test case generation

We adopted random test case generation to generate the source test case. These randomly generated inputs were not embedded with any domain-specific knowledge—in other words, we did not rely on any semantically meaningful data [18]. This is consistent with the fundamental philosophy of metamorphic testing, which focuses on the relations between multiple inputs and their corresponding outputs,

Table 4
Mutation operators covered by selected mutants.

MO_{prog}	Description
AOIS	Replacing Increment or Decrement Operators
AORB	Replacing Binary Arithmetic Operators
ROR	Replacing Relational Operators
VDL	Deleting or Replacing Variables
ODL	Deleting or Replacing Operands
ASRS	Replacing Array Subscripts or Shifts

rather than the correctness of individual outputs. In fact, random data may even be more effective in revealing faults [28].

Ultimately, we generated 30 source test cases for each indicator system. Based on these source test cases, we applied 9 MRs to each one, resulting in a total of 270 source/follow-up input pairs (30 source test cases × 9 MRs). All these test pairs were executed on the indicator systems to check for any violations of the metamorphic relations. To improve the reliability of our results, each experiment was repeated 10 times.

4.5. Generation of mutants

For mutation testing, we used MuJava [29] to systematically generate mutants for the indicator system. MuJava is a powerful automated mutation analysis system that allows users to select relevant source files for mutation. We introduced faults into each case's code implementation using MuJava. Subsequently, we manually screened for equivalent mutants and removed non-compilable versions. Ultimately, we selected 21 valid mutants for SQ, 35 for Wine, and 29 for Eco. The mutation operators(MO_{prog}) covered by these mutants are summarized in Table 4. We systematically validated whether each metamorphic relation was violated by the generated mutants. A mutant was considered "killed" when its execution results violated any predefined metamorphic relation, thereby indicating the successful detection of an error in the mutant.

4.6. Evaluation metrics

In our experiments, we selected two metrics to evaluate the effectiveness of ISVMT: the mutation score(MS) and the metamorphic relation violation percentage(VP). The calculation method for the variation score is provided in Eq. (4).

$$MS(MUT, TS) = \frac{|MUT_{\text{killed}}|}{|MUT|} \tag{4}$$

A metamorphic relation is considered violated if the source and follow-up test cases fail to satisfy the expected relationship. The metamorphic violation percentage is defined as the ratio of the number of test case pairs violating the relationship to the total number of test case pairs evaluated. Eq. (5) gives the formula for the percentage of metamorphic relation violation, Where $|TC|$ denotes the total number of test cases, $|TC_{\text{violated}}|$ represents the number of test cases that violate the metamorphic relation.

$$VP = \frac{|TC_{\text{violated}}|}{|TC|} \tag{5}$$

4.7. Baseline

Xu et al. [30] proposed SuperCLUE, a benchmark system designed for evaluating large language models. To validate the effectiveness of its evaluation results, the researchers compared the scores assigned by SuperCLUE with human ratings, and computed the correlation between them. A higher correlation indicates that the evaluation results are closer to human judgment, while a lower correlation suggests potential fault in the evaluation system.

Table 5
Validation results on the original dataset.

MR	Eco		SQ		Wine	
	NP	VP	NP	VP	NP	VP
ZIR	Y	0	Y	0	Y	0
SIR	Y	0		\	Y	0
CR	Y	**0.21**	Y	0	Y	0
RS		0.32		\		0.46
RIR	Y	0	Y	0	Y	0
ISIR	Y	0	Y	0	Y	0
MAXR	Y	**0.73**		\	Y	0
MINR	Y	**0.60**		\	Y	0
WIR		1.0	Y	0		1.0

Inspired by their methodology, we use it as a baseline for our study. Specifically, we compare the outputs of the indicator system with the human-labeled ground truth results (Test Oracle) and calculate the correlation between them. A low correlation means that the evaluation results deviate from human expectations, indicating a potential fault in the indicator system.

Taking the wine quality assessment indicator system as an example, we use a public dataset[2] containing multiple wine samples, each annotated with a quality score assigned by professional tasters. Each sample is denoted as $x_i (i = 1, 2, \ldots, n)$, where y_i represents the expert-assigned quality score (i.e., the Test Oracle), and $\hat{y}_i$ denotes the score generated by the indicator system. To validate the system's effectiveness, we compute the Pearson Correlation Coefficient r between y and $\hat{y}$. A coefficient r approaching 1 indicates high consistency with expert scores, while a low coefficient(< 0.5) suggests potential fault.

In the mutation analysis, we generate k mutants and apply the dataset to each mutant j and obtain its predicted output $\hat{y}^{(j)}, j = (1, 2, \ldots, k)$. We then calculate the correlation coefficient r between $\hat{y}^{(j)}$ and y. A mutant is considered killed if the correlation coefficient falls below a threshold of **0.5**, indicating that the mutant produces results significantly deviating from the expected values. Finally, we compare the mutation score derived from this correlation-based method with ISVMT.

5. Result analysis

5.1. RQ1: effective of ISVMT

The results are presented in Table 5. Our MRs are proposed based on domain knowledge and user expectations, rather than being tied to the specific implementation of any indicator system. When applying MRs to validate a particular indicator system, we first analyze whether each relation is a necessary property of the given algorithm implementation. If an MR represents a necessary property of that system, the column "NP" (Necessary Property) is marked as "Y". The column "VP" indicates the violation percentage observed for each MR.

Clearly, when validating an MR marked as NP, a non-zero VP indicates a fault. On the other hand, if an MR is not marked as NP but still yields a non-zero VP, it suggests a discrepancy between the actual behavior of the indicator system and the general expectations users typically have [18]. This deviation implies that the current algorithm may not be suitable for users who expect the system to satisfy the corresponding MR.

[2] https://www.datacastle.cn/dataset_description.html?id=756&type=dataset

5.1.1. *Eco*

1. Analysis of necessary properties violations

The experimental results for the Eco revealed violations of three necessary properties (NP), corresponding to the metamorphic relations “CR”, “MAXR”, and “MINR”. These findings provide strong evidence that the implementation of this indicator system contains faults. The CR relation states that for a positive indicator (where a higher value indicates better performance), if a sample’s value for that indicator increases, its evaluation result should increase accordingly, while the results of other samples should remain unchanged. However, experimental data showed that 20.7% of the source/follow-up test case pairs violated this relation. The MAXR relation requires that when a new sample is added to the source test case, with each of its indicator values being the best among all existing samples (i.e., maximum for positive indicators and minimum for negative indicators), the new sample’s evaluation result should be better than all existing ones. In the experiment, 73.3% of the test case pairs failed to meet this requirement. The MINR relation also showed significant violations, with a violation rate of 60.0%.

2. Analysis of nonessential properties violations

In addition, violations were observed for some properties that were not marked as necessary (non-NP). These violations suggest that the actual behavior of the indicator system deviates from users’ general expectations, as expressed through the corresponding MRs. For each such deviation, we analyzed the specific reasons behind the violations.

RS. RS relation assumes that if a sample is removed without affecting the attribute information of the remaining samples, then the evaluation results of those remaining samples should remain unchanged. However, the experimental results show that 31.7% of the source/follow-up test case pairs violated this relation. Upon analyzing the implementation of the Eco indicator system, we found that Eco adopts max normalization to address unit differences among evaluation indicators. The normalization process is defined as follows:

$$x'_{ij} = \frac{x_{ij}}{\max_{1\le i\le m}(x_{ij})}$$

Here, x_{ij} denotes the value of sample i on indicator j, and x'_{ij} is the normalized value. This method introduces a global dependency, as removing a sample with the maximum value in a given indicator will change the denominator, thus affecting the normalized values of all remaining samples. This is the root cause of the observed violations of the RS relation. If the user has the same expectation as RS, i.e., the expectation that each sample’s computed result relates only to its own original value and is not affected by the presence or change of other samples (i.e., the samples are independent of each other), then maximum value normalization is not suitable for this scenario. In the Wine case, 45.7% of the source/follow-up test case pairs violated the RS metamorphic relation, which was also caused by the adopted data standardization method.

Data standardization can be beneficial in indicator systems, as it allows for comparability across indicators with different scales. Common standardization approaches include min–max normalization, z-score standardization, decimal scaling, and log transformation, among others. However, this diversity of standardization techniques also raises important concerns. These transformations essentially perform a mapping of the original data into another space, which can influence how indicator systems behave under certain data modifications.

Our proposed MRs are not intended to criticize standardization itself. Rather, they serve as a tool to verify whether the behavior of an indicator system—after such data mapping—still aligns with expected human intuitions or properties. For example, if removing a single sample causes a disproportionately large change in evaluation results, it may indicate that the standardization method introduces instability or sensitivity, which practitioners should be aware of. In this sense, violations of certain MRs—which are not necessarily inherent properties of the indicator system’s algorithmic implementation—do not always indicate defects in the algorithm. Instead, they highlight behavioral deviations that might be counterintuitive depending on the context. Therefore, our goal is to support users and developers in analyzing whether the chosen standardization approach is appropriate for their specific evaluation task.

WIR. In an indicator system, weights are used to assign importance to different indicators by calculating $x_i \times w_i$. The WIR (Weight Increase Relation) metamorphic relation assumes that when the weight wi of a positive indicator increases—while all other weights remain unchanged—the value of $x_i \times w_i$ will also increase. As a result, the overall evaluation score of a sample is expected to rise. Moreover, samples with higher values for that indicator are expected to show a greater increase in their evaluation results. However, our experimental results showed a 100% violation of this relation. To investigate this, we closely examined the internal logic of the Eco and found that it adopts a relative scoring mechanism. Specifically, the algorithm first selects an ideal sample—a hypothetical sample composed of the maximum values across all indicators—and then calculates the Euclidean distance between each sample and this ideal one. The closer a sample is to the ideal point, the better its evaluation result; conversely, the farther away it is, the worse its score.

When we increase the weight of a certain indicator, it effectively amplifies the differences along that dimension in the distance calculation. This causes the distances between samples and the ideal sample to become larger overall, which in turn leads to a decrease in evaluation scores—contrary to the expected outcome based on WIR.

5.1.2. *SQ*

All test case results conformed to the defined MRs, and our approach did not detect any issues in the SQ. The reliability of this verification still requires further testing in the second phase of our study.

We marked SIR, RS, MAXR, and MINR as “\” because these MRs were not applicable to this particular indicator system. Specifically, the input of this system consists of manual scores assigned to each indicator, with identical value ranges and no differences in units or scales, making SIR (Scale Invariance Relation) inapplicable. Moreover, this indicator system evaluates only one sample at a time, which prevents the application of RS, MAXR, and MINR relations.

5.1.3. *Wine*

1. Analysis of necessary properties violations

All test case results conformed to the defined MRs, and our approach did not detect any fault in the Wine. The reliability of this verification still requires further testing in the second phase of our study.

2. Analysis of nonessential properties violations

For the non-essential properties, we observed two violations, indicating that the actual behavior of the indicator system deviated from the corresponding expectations. RS. The violation of the RS (Remove Sample) relation occurred for the same reason as in the Eco.

WIR. The Wine system adopts z-score standardization, using the formula:

$$Z_i = \frac{X_i - \mu}{\sigma}$$

where μ represents the mean and σ the standard deviation of each indicator. The standardized score Z_i reflects the relative position of a sample: a positive value indicates performance above the mean, while a negative value indicates the opposite.

Since the final evaluation score is typically computed as a weighted sum of standardized indicator values (e.g., $\sum w_j z_{ij}$), increasing the weight w_j of a particular indicator may actually decrease a sample’s overall score, if its original value X_i is below the mean. In such cases, the negative contribution of that indicator is magnified, resulting in a lower total score.

Z-score standardization transforms each indicator value into its deviation from the mean, making the resulting values reflect relative positions rather than absolute performance. As a result, it does not guarantee a monotonic positive correlation between indicator weights and final evaluation scores. If a sample's raw value on a certain indicator is below the average, increasing the weight of that indicator can lead to a decrease in the evaluation result, contrary to common intuition.

Therefore, in applications where interpretability of weights and a linear, intuitive response of evaluation results to weight changes are important, the use of z-score standardization is not recommended, as it may introduce such non-monotonic and counterintuitive behaviors.

5.2. RQ2: How effective is ISVMT on mutated datasets?

In RQ1, we used the first phase of ISVMT to detect faults in the indicator systems and to identify deviations from expected behavior. For the SQ indicator system, no violations were observed. To further investigate the effectiveness of the validation process, in this section we apply Phase II, where artificial faults (mutants) are injected into the code implementation of the indicator systems to assess whether our method can detect them.

In this phase, we used the source/follow-up test case pairs generated during Phase I to demonstrate the effectiveness of the initial validation. A mutant is considered "killed" if at least one of its test case pairs violates an expected metamorphic relation.

Since the goal of mutation analysis in this context is verification, we only apply essential metamorphic relations—i.e., those that represent necessary properties of the indicator system. For each such relation, if violations are observed in any mutant, we consider that mutant as killed by the corresponding MR, indicating that a fault has been successfully detected. The purpose of the experiment is to compute the percentage of mutants killed by each MR, which serves as a measure of the fault detection effectiveness of our approach.

To comprehensively evaluate the performance of the proposed general MRs, we meticulously recorded the violation percentages for each MR. Specifically, a MR is considered violated when the source test case and the follow-up test case do not satisfy that relation. Accordingly, Table 6 to Table 8 present detailed information on the detection efficiency of various MRs across three different cases. The first column lists different versions of the mutants, where "Original" refers to the version without any injected faults. Each subsequent row (except the last two) shows the violation percentage (VP) of each MR for the corresponding mutant. The Total row summarizes the number of mutants killed by each MR, while the Ms row records the overall mutation score.

Eco Since we identified real defects in the Eco indicator system during the validation in Phase I, injecting additional artificial faults would be redundant. To demonstrate the effectiveness of our approach, we first corrected the identified issues and then conducted mutation analysis. As shown in the "Original" row, the repaired indicator system did not violate any metamorphic relations, indicating that the fixes were successful.

For each mutant, if any MR is violated, the mutant is considered to be killed. The final mutation score is then calculated. According to Table 6, ISVMT demonstrates strong effectiveness on the Eco indicator system, achieving a high mutation score of 0.93. This further validates the effectiveness of our approach and indicates that both the selected metamorphic relations and the generated test cases are of high quality, enhancing our confidence in the Phase I validation results.

In addition, our method reveals significant differences in the fault-detection capabilities of different MRs. For example, CR killed 26 out of 29 mutants, demonstrating strong effectiveness, whereas ISIR only killed 10 out of 29, indicating relatively lower detection capability.

Wine Table 7 presents the effectiveness of different metamorphic relations (MRs) in detecting mutants in the Wine case study. Each cell value represents the Violation Percentage (VP) of a mutant under a specific MR, indicating the fault-detection capability of that MR.

As shown in the Total row, CR and RIR demonstrate the strongest detection capabilities, each successfully killing 15 mutants. In contrast, MINR shows weaker performance, killing only 8 mutants. Overall,

Table 6
Effectiveness of metamorphic relations for Eco.

Mutant	MRs						
	ZIR	SIR	CR	RIR	ISIR	MAXR	MINR
Original	0.00	0.00	0.00	0.00	0.00	0.00	0.00
v1	0.00	0.00	0.00	0.00	0.00	0.00	0.00
v2	0.73	0.73	0.74	0.85	0.73	0.90	0.75
v3	0.00	0.00	0.10	0.51	0.00	0.47	0.50
v4	0.53	0.53	0.58	0.73	0.53	1.00	1.00
v5	0.00	0.19	0.00	0.81	0.00	0.00	0.18
v6	0.00	0.15	0.00	0.84	0.00	0.00	0.45
v7	0.53	0.53	0.60	0.93	0.53	1.00	1.00
v8	1.00	1.00	1.00	0.00	1.00	1.00	0.09
v9	1.00	1.00	1.00	1.00	1.00	1.00	1.00
v10	1.00	0.00	0.01	0.00	0.00	0.00	0.00
v11	1.00	0.00	0.53	0.00	0.56	0.00	0.00
v12	1.00	1.00	1.00	1.00	1.00	1.00	1.00
v13	1.00	1.00	1.00	1.00	1.00	1.00	1.00
v14	1.00	1.00	1.00	0.00	1.00	1.00	0.09
v15	0.00	0.00	0.45	0.48	0.00	0.53	0.96
v16	0.00	0.00	0.05	0.00	0.00	0.00	0.12
v17	0.81	0.30	0.80	0.92	0.30	1.00	1.00
v18	0.00	0.00	0.00	0.00	0.00	0.00	0.00
v19	0.00	0.00	0.42	0.00	0.00	1.00	0.39
v20	0.00	0.00	0.21	0.00	0.00	0.30	0.03
v21	0.00	0.00	0.29	0.00	0.00	0.00	0.00
v22	0.00	0.00	0.01	0.00	0.00	0.00	0.00
v23	0.00	0.00	0.08	0.00	0.00	1.00	1.00
v24	0.00	0.00	0.02	0.00	0.00	0.00	0.00
v25	0.00	0.00	1.00	0.00	0.00	0.00	0.91
v26	0.00	0.00	0.36	0.00	0.00	0.50	0.05
v27	0.00	0.00	0.34	0.00	0.00	0.03	0.12
v28	0.00	0.00	0.10	0.00	0.00	0.03	0.22
v29	0.00	0.00	1.00	0.00	0.00	0.00	0.91
Total	11	11	26	11	10	17	22
MS	0.93						

the Mutation Score (MS) for this case is 0.83, indicating a generally effective testing outcome.

SQ For the SQ, the mutation score MS is 1.0, indicating that all mutants were detected by at least one MR, which demonstrates the high effectiveness of the ISVMT. Among all the MRs, the CR performed best, killing 21 mutants. This was followed by RIR, WIR, ZIR, and ISIR, which killed 6, 5, 5, and 6 mutants, respectively. This suggests that CR is the most effective MR and can serve as a priority for test case design.

All values in the Original row are 0, indicating that none of the MRs were violated in the original version, which further confirms the soundness and validity of the defined metamorphic relations.

Overall, our method demonstrates significant differences in the fault-detection capabilities of various metamorphic relations (MRs) during mutation analysis. These variations are closely related to the underlying logic and internal consistency of the indicator systems. Notably, in all three case studies, the CR (Consistency Relation) consistently achieved the highest fault-detection rate, killing 26/29, 24/35, and 21/21 mutants, respectively. In contrast, in the SQ case, the WIR (Weight Influence Relation) showed the weakest performance, detecting only 5 out of 21 mutants, indicating its relatively low effectiveness.

Moreover, the presence of surviving mutants can be attributed to the limited coverage of current MRs, which may not fully encapsulate all critical properties of the system, resulting in test cases failing to expose certain faults. This highlights the need for more fine-grained and targeted MRs. Additionally, although the number of test cases was sufficient, the diversity of their design might be inadequate to cover all potential fault scenarios. These factors collectively contribute to the existence of surviving mutants and underscore the importance of enhancing both the comprehensiveness of MRs and the diversity of test case design in future work.

Table 7
Effectiveness of metamorphic relations for Wine.

Mutant	MRs					
	ZIR	CR	SIR	MAXR	MINR	RIR
Original	0.00	0.00	0.00	0.00	0.00	0.00
v1	1.00	1.00	1.00	1.00	1.00	1.00
v2	0.00	0.00	0.00	0.00	0.00	0.00
v3	0.05	0.71	0.08	0.00	0.00	0.93
v4	0.06	0.04	0.05	0.90	0.93	0.11
v5	1.00	1.00	1.00	1.00	1.00	1.00
v6	1.00	1.00	1.00	1.00	1.00	1.00
v7	1.00	1.00	1.00	1.00	1.00	1.00
v8	0.00	0.00	0.00	0.00	0.00	0.15
v9	0.00	0.00	0.00	0.00	0.00	0.15
v10	0.00	0.00	0.00	0.00	0.00	0.00
v11	0.00	0.00	0.00	0.00	0.00	0.00
v12	0.00	0.00	0.00	0.00	0.00	0.00
v13	0.00	0.02	0.00	0.00	0.00	0.00
v14	0.00	0.00	0.00	0.00	0.00	0.00
v15	1.00	1.00	1.00	1.00	1.00	1.00
v16	1.00	1.00	1.00	1.00	1.00	1.00
v17	0.00	0.00	0.00	0.00	0.00	0.13
v18	0.00	0.00	0.00	0.00	0.00	0.10
v19	0.00	1.00	0.00	1.00	0.00	0.00
v20	0.00	0.00	0.00	0.00	0.00	0.00
v21	0.00	0.42	0.77	0.00	0.00	0.78
v22	0.00	1.00	0.00	1.00	0.00	0.00
v23	0.00	0.00	0.00	0.00	0.00	0.00
v24	0.00	0.00	0.00	0.00	0.00	0.00
v25	0.00	0.10	0.00	0.00	0.00	0.00
v26	0.89	0.15	1.00	0.00	0.00	1.00
v27	0.00	1.00	0.00	1.00	0.00	0.00
v28	0.00	0.55	0.00	0.00	0.00	0.00
v29	0.00	0.00	1.00	0.00	0.00	0.00
v30	0.08	0.71	0.05	0.00	0.00	0.95
v31	0.00	1.00	0.00	1.00	1.00	0.00
v32	0.00	1.00	0.00	1.00	0.00	0.00
v33	0.00	1.00	0.00	1.00	0.00	0.00
v35	0.00	1.00	0.00	1.00	0.00	0.00
v35	0.00	1.00	0.00	1.00	0.00	0.00
Total	10	24	12	15	8	15
MS	0.83					

Table 8
Effectiveness of metamorphic relations for SQ.

Mutant	MRs				
	ZIR	CR	RIR	ISIR	WIR
Original	0.00	0.00	0.00	0.00	0.00
v1	0.00	0.28	0.00	0.31	0.00
v2	0.00	0.13	0.00	0.00	0.00
v3	0.00	0.41	0.00	0.46	0.00
v4	0.37	0.75	0.37	0.76	0.37
v5	0.00	0.13	0.80	0.00	0.00
v6	0.70	0.79	0.70	0.89	0.70
v7	0.19	0.11	0.00	0.00	1.00
v8	0.00	0.66	0.00	0.00	0.00
v9	0.00	0.63	0.00	0.00	0.00
v10	0.00	0.61	0.00	0.00	1.00
v11	0.00	0.70	0.00	0.00	0.00
v12	0.00	0.64	0.00	0.00	0.00
v13	0.44	0.11	0.00	0.00	0.00
v14	0.00	0.61	0.00	0.00	0.00
v15	0.00	0.23	0.25	0.00	0.00
v16	0.00	0.19	0.21	0.00	0.00
v17	0.00	0.10	0.00	0.00	0.00
v18	0.35	0.17	0.03	0.33	0.03
v19	0.00	0.08	0.00	0.00	0.00
v20	0.00	0.36	0.00	0.00	0.00
v21	0.00	0.14	0.00	0.00	0.00
Total	5	21	6	5	5
MS	1.0				

5.3. RQ3: How effective is ISVMT compared to other validation methods?

Since the method of statistical analysis relies on a manually labeled dataset as Test Oracle, and the relevant datasets were not provided for SQ and Eco, we performed the experimental analysis on Wine only. The Wine dataset contains approximately 1,600 samples of different wines, each annotated with a manually labeled quality score as the ground truth. We conducted three experiments by randomly selecting 50, 500, and 1,000 samples from the dataset, respectively. In each experiment, every mutant was used to evaluate the selected samples, generating predicted scores. We then calculated the correlation coefficient between the predicted results and the true labels. As described in the baseline method in Section 4.7, if the correlation coefficient is below 0.5, the mutant is considered to be successfully killed. Finally, we calculated the mutation score based on the proportion of mutants that were killed.

Table 9 demonstrates the effectiveness of the statistical analysis method in mutation detection. Each row (except for the last two) records the Pearson Correlation Coefficient (pcc) and the Spearman Rank Correlation Coefficient (srcc) for different mutant versions under varying sample sizes. Correlation values that are significantly low (< 0.5) are highlighted. If an error in a mutant leads to a failure in correlation computation (as correlation requires vectors of equal length), the corresponding cell is marked as "NA", and the mutant is considered killed.

We calculated the correlation coefficients using the Java Math package. The last two rows of the table display the total number of killed mutants and the mutation score. The highest result is 0.65. However, this is lower than the mutation score (0.83) achieved by ISVMT.

In some indicator systems, mutations may only lead to minor changes in the output, resulting in correlation coefficients that remain relatively high. In such cases, correlation analysis struggles to distinguish mutated versions from the original, thereby leading to a lower mutation kill rate. Therefore, although correlation analysis is a general-purpose statistical tool with some reference value in certain contexts, it performs significantly worse than the metamorphic testing–based ISVMT method when applied to the validation of indicator systems.

In addition, our method does not rely on manually labeled data as a Test Oracle, which alleviates the problem of missing Test Oracles in the validation of the indicator system. This improves validation efficiency compared to traditional methods.

5.4. RQ4: How does ISVMT compare to LLMs in terms of effectiveness?

In RQ4, we evaluate the verification effectiveness of ISVMT and LLMs by comparing their mutation scores on the mutated dataset. To the best of our knowledge, there are currently no mature methods that directly apply LLMs to the task of indicator system verification. Considering the effectiveness large language models have demonstrated in code comprehension and test generation tasks, we directly use prompting LLMs as a baseline method for our experiment. Specifically, we employ the GPT-3.5-turbo, QWen-corder-turbo and DeepSeek-V3 model, providing the description of the indicator system and the code snippet with injected errors as input, and directly prompting the model to determine whether there are errors in the indicator system. Subsequently, we manually check whether the LLM's response aligns with the injected errors: if the LLM accurately identifies and points out the injected errors, the mutant is considered successfully killed; if the LLM fails to identify the injected errors and instead provides suggestions related to code style, comments, or other irrelevant issues, it is considered that the LLM failed to detect the injected errors.

Considering that LLMs may mistakenly identify correct code as defective, we first conducted an evaluation on the original versions of the code. For the SQ and Wine cases, all three methods only provided suggestions related to code style, comments, and robustness improvements, without identifying any actual functional errors. Therefore, we believe the LLMs considered the code to be correct. This aligns with our

Table 9
Validation results of statistical analysis methods.

Mutant	50		500		1000	
	srcc	pcc	srcc	pcc	srcc	pcc
original	0.6420	0.5466	0.5771	0.5238	0.5735	0.5274
v1	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**
v2	0.6416	0.5462	0.5771	0.5233	0.5735	0.5276
v3	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**
v4	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**
v5	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**
v6	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**
v7	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**
v8	0.6416	0.5462	0.5771	0.5233	0.5735	0.5276
v9	0.6416	0.5462	0.5771	0.5233	0.5734	0.5276
v10	**0.4985**	**0.3657**	**0.3355**	**0.2443**	**0.3653**	**0.2829**
v11	0.5420	**0.4030**	**0.3724**	**0.2765**	**0.3926**	**0.3071**
v12	0.5420	**0.4030**	**0.3724**	**0.2765**	**0.3926**	**0.3071**
v13	**0.3464**	**0.1948**	0.5801	**NA**	0.5763	**NA**
v14	0.6416	0.5462	0.5771	0.5233	0.5735	0.5276
v15	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**
v16	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**
v17	0.6467	0.5459	0.5770	0.5233	0.5734	0.5276
v18	0.6436	0.5466	0.5771	0.5234	0.5735	0.5278
v19	0.6354	**NA**	0.5801	**NA**	0.5763	**NA**
v20	0.5007	**0.3659**	**0.3357**	**0.2444**	**0.3654**	**0.2829**
v21	**0.3544**	**0.2971**	**0.3182**	**0.1895**	**0.4073**	**0.2949**
v22	0.6354	**NA**	0.5801	**NA**	0.5763	**NA**
v23	0.6434	0.5466	0.5758	0.5231	0.5741	0.5269
v24	0.6416	0.5462	0.5771	0.5233	0.5735	0.5276
v25	0.6416	0.5462	0.5769	0.5232	0.5735	0.5276
v26	0.6416	0.5462	0.5771	0.5233	0.5735	0.5276
v27	0.6354	**NA**	0.5801	**NA**	0.5763	**NA**
v28	0.6416	0.5462	0.5771	0.5233	0.5735	0.5276
v29	0.6416	0.5462	0.5771	0.5233	0.5735	0.5276
v30	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**
v31	**−0.3705**	**−0.5462**	**−0.4169**	**−0.5233**	**−0.4209**	**−0.5276**
v32	0.6354	**NA**	0.5801	**NA**	0.5763	**NA**
v33	0.6354	**NA**	0.5801	**NA**	0.5763	**NA**
v34	0.6354	**NA**	0.5801	**NA**	0.5763	**NA**
v35	0.6354	**NA**	0.5801	**NA**	0.5763	**NA**
total	13	23	15	23	15	23
MS	0.37	0.65	0.43	0.65	0.43	0.65

Table 10
Compare with LLMs in terms of effectiveness.

Method	SQ	Wine	Eco
GPT-3.5-turbo	0.67	0.75	0.52
QWen-corder-turbo	0.71	0.82	0.69
DeepSeek-V3	0.90	**1.0**	0.76
ISVMT	**1.0**	0.83	**0.93**

validation results from the first phase for these two indicator systems. In contrast, for the Eco case, the ISVMT method successfully detected a real defect, whereas the other two methods failed to identify the issue.

Then we calculate the mutation score based on the LLM's feedback of mutants, with the results shown in Table 10.

The three indicator systems vary significantly in computational complexity, which in turn impacts the fault detection performance of both large language models (LLMs) and ISVMT. The Wine system employs a simple normalization and weighted summation method based on linear operations. Due to its straightforward logic, all three LLMs performed well on this dataset: GPT achieved a mutation score of 0.75, QWen and DeepSeek showed similar trends, with DeepSeek reaching a perfect score of 1.0, exceeding ISVMT's 0.83. In contrast, the SQ system relies on fuzzy comprehensive evaluation, involving nonlinear membership functions and fuzzy transformations. The added complexity led to a performance drop for GPT-3.5, which scored only 0.67, while ISVMT maintained a perfect score of 1.0. The Eco system, based on the TOPSIS method, includes entropy weighting and Euclidean distance calculations, representing the highest complexity among the three. Here, LLMs again struggled: GPT scored 0.52, QWen 0.69, and DeepSeek 0.76, whereas ISVMT achieved 0.93.

Overall, the three LLMs exhibit notable performance differences as the complexity of the indicator systems increases. On the simplest system, Wine, LLMs outperform the other two systems and even surpass ISVMT; however, their performance declines on more complex systems. We believe this is because LLMs find it easier to understand the entire computation process in simple logical scenarios, enabling them to quickly detect potential anomalies or errors in the indicator system's code. When faced with complex logic and nonlinear computations, however, LLMs tend to focus only on superficial code issues and fail to accurately capture the reasoning behind the evaluation or identify potential defects. In contrast, ISVMT demonstrates significant effectiveness in handling highly complex models. Yet, on simple logical indicator systems, ISVMT's performance is inferior to that of LLMs, likely due to the limited design space for metamorphic relations restricting its fault detection capabilities . Additionally, the variation in mutant difficulty—introduced through random mutation using Mujava—may also contribute to the observed differences in detection performance.

In the current experiment, the prompts used primarily instructed the LLM to directly determine whether errors existed in the indicator system, making the task relatively ambiguous. The prompts did not guide the LLM to delve into the computational logic of the indicator systems, such as the specific steps involved in fuzzy comprehensive evaluation or TOPSIS. Consequently, the model tended to focus on surface-level issues like code style and comments rather than identifying logical errors. For complex indicator systems like SQ and Eco, which involve multi-step calculations, prompts that failed to break down the task or guide the LLM through step-by-step reasoning likely hindered the model's ability to grasp the full logic in a single inference, leading to reduced error detection performance. Future research could explore enhancing prompts, providing more contextual information, and employing step-by-step reasoning to better guide the LLM in understanding intricate logic. Additionally, combining LLMs with MT and automated testing tools could further unlock their potential in verifying indicator systems, improving their capacity to detect errors in complex computational processes.

6. Discussions and lessons learned

6.1. MT on indicator systems VS machine-learning models

MT has been widely adopted for machine-learning (ML) models, the application to indicator systems introduces three novel aspects that go beyond existing ML-oriented MT literature.

Domain-Invariant MRs vs. Model-Specific MRs. ML-focused MT typically tailors metamorphic relations (MRs) to the model architecture or learning objective (e.g., input perturbation, label flipping, or dropout invariance). In contrast, indicator system MRs are derived from domain-agnostic logical properties (e.g., monotonicity, unit invariance, or rank preservation) that are independent of the underlying algorithm. This allows ISVMT to validate any indicator system—whether fuzzy, statistical, or neural—without reverse-engineering its internals.

Data-Mutation-Driven Test Generation. Traditional ML-MT often relies on model-specific data augmentation (e.g., image rotation, synonym replacement). ISVMT introduces data mutation operators that systematically alter indicator values while preserving domain semantics (e.g., scaling a "green-sales rate" metric or zero-weighting a sub-indicator). This is distinct from ML-MT's focus on preserving label semantics.

Program-Mutation Validation Layer. ML-MT rarely evaluates the robustness of the testing method itself. ISVMT uniquely augments MT with indicator system mutation (Section 3.3) to inject realistic faults (e.g., formula errors, weight miscalculations). This dual-layer validation—MR violation + mutation score—ensures that both the system and the testing method are stress-tested, an approach not seen in prior ML-MT work.

ISVMT generalizes MT from validating model behavior to validating domain-logic consistency across diverse indicator systems, providing a model-agnostic, domain-invariant validation framework.

6.2. Limitations and future work

In the application of metamorphic testing, there are two critical steps: the identification of metamorphic relations (MRs) and the generation of test cases [31].

Regarding the identification of metamorphic relations, this study proposed nine MRs specifically tailored to indicator systems. These relations rely heavily on domain knowledge. However, due to the limitations of our knowledge and experience, the proposed MRs may not fully cover all functional aspects of the systems under test. The existence of surviving mutants in our mutation analysis supports this concern. Moreover, experimental results show significant variation in the fault detection effectiveness of different MRs. In future work, we aim to design metamorphic relations with stronger fault detection capabilities to enhance the overall effectiveness of ISVMT.

In addition, the identification of MRs is a labor-intensive process that requires a deep understanding of the domain. To reduce manual effort and improve scalability, future research should explore the automated generation of metamorphic relations [32,33]. One promising direction is to leverage domain ontologies or user requirements, or to utilize large language models (LLMs) to extract metamorphic relations from natural language descriptions provided by users.

In terms of test case generation, our current approach employs random generation of source test cases. While this method provides a degree of representativeness, previous studies have shown that more targeted generation of source and follow-up test cases can significantly improve the effectiveness of metamorphic testing [31,34]. Therefore, we plan to explore smarter or heuristic-based test case generation techniques in future work, in order to further enhance the fault detection capability and applicability of ISVMT.

7. Related work

7.1. MT

MT is a method that encompasses both the creation of test cases and the validation of test outcomes. It revolves around a core component known as metamorphic relations, which are essential characteristics of the subject function or algorithm concerning various inputs and the anticipated outputs associated with them [17]. MT has currently achieved good results in many fields. For instance, Xie et al. [18] present a technique for testing the implementations of machine learning classification based on the MT. Jiang et al. [35] address a crucial issue, specifically false satisfactions—instances where metamorphic relations are satisfied despite at least one failing execution—and re-examine the use of MT (MT) in the context of sentiment analysis (SA) systems. Xie et al. [36] have devised a MT methodology for the assessment and validation of unsupervised machine learning systems. Chaleshtari et al. [37] introduce Metamorphic Security Testing for Web-interactions (MST-wi), an innovative MT methodology. This approach incorporates test input generation techniques inspired by mutational fuzzing, effectively addressing the oracle problem commonly encountered in security testing. Zhou et al. [38] have expanded MT to develop a user-centric methodology for software verification, validation, and quality evaluation. Sun et al. [39] propose improving the cost-effectiveness of MT by leveraging the feedback information obtained in the test execution process. Luu et al. [40] identify inherent mathematical properties of linear regression, subsequently proposing 11 Metamorphic Relations for application in testing scenarios. Xiao et al. [41] presents MT-DLComp, a specialized MT framework tailored for deep learning (DL) compilers, aimed at efficiently detecting faulty compilations. The methodology employs carefully crafted metamorphic relations (MRs) to initiate semantics-preserving mutations on deep neural network (DNN) models, thereby generating their variants for testing purposes. Deng et al. [20] introduce an innovative declarative rule-based MT framework, termed RMT. This framework offers a rule template with natural language syntax, enabling users to flexibly define a diverse array of testing scenarios grounded in real-world traffic rules and domain expertise. Luu et al. [42] present the Sequential MT (SMART) framework, an extension of a highly successful MT methodology from the software testing domain, aimed at addressing this challenge. The framework employs sequences of metamorphic test case groups to ascertain the accuracy of autonomous vehicle (AV) decision-making. Paltenghi et al. [43] introduce MorphQ, marking it as the pioneering MT approach tailored for quantum computing platforms. Ayerdi et al. [44] propose an MR pattern, termed PV, designed for the identification of performance-driven metamorphic relations. We demonstrate its applicability across two distinct CPS domains: automated navigation systems and elevator control systems.

7.2. Mutation testing

Mutation testing stands as the most recognized standard within the fault injection testing methodology, serving to assess and enhance the robustness of a suite of test cases [45]. Mutation testing is a very effective testing technology. Petrovi et al. [46] analyze offer further substantiation that mutants are indeed associated with genuine faults. In addition, Petrovi et al. [47] introduce a scalable strategy for mutation testing. Kaufman et al. [48] present an innovative measure for evaluating the usefulness of mutants, known as the test completeness advancement probability (TCAP). Dakhel et al. [49] present MuTAP (Mutation Test case generation using Augmented Prompt), a method designed to enhance the efficacy of test cases produced by Large Language Models (LLMs) in uncovering bugs, achieved through the application of mutation testing principles. Degiovanni et al. [50] introduce μBert, a mutation testing tool that leverages a pre-trained language model, specifically CodeBERT, to generate mutants. Sánchez et al. [51] present the findings of an in-depth investigation into the application of mutation testing within GitHub projects. Fortunato et al. [52] introduce an innovative suite of mutation operators designed to create mutants by leveraging qubit measurements and quantum gates. Sun et al. [53] suggest the implementation of five specialized mutation operators aimed at tackling vulnerabilities by assessing the adequacy of testing in Ethereum Smart Contracts. Delamaro et al. [54] introduce an interprocedural mutation-based criterion, termed Interface Mutation (IM), which is specifically designed for application in the context of integration testing.

7.3. Software validation

Software validation is a critical process in the software development lifecycle that ensures the software product meets the specified requirements and fulfills its intended purpose. It involves a series of activities designed to confirm that the software behaves as expected and satisfies the needs of its users and stakeholders [55]. Software validation is applicable across a wide range of software domains, each with its specific requirements and challenges. There are some key areas where software validation plays a crucial role. For example, Basili et al. [56] report on a study where they conducted an empirical examination of the set of object-oriented (OO) design metrics. Gyimothy et al. [57] outline the methodology employed to compute the object-oriented metrics proposed by Chidamber and Kemerer, demonstrating their application in assessing the fault-proneness of the source code for the open-source Web and email suite, Mozilla. Park et al. [58] present an innovative, analytics-driven approach to user-assisted software validation for quantum neural network (QNN) codes. Rouland et al. [59] propose an integrated model-driven approach for vulnerability detection and treatment during software architecture design.

Febrero et al. [7] introduced an innovative approach to software reliability analysis by leveraging a structural model grounded in widely-recognized international standards, tailored specifically for industry applications. Gao et al. [3] proposed a software quality assessment model by examining the intrinsic qualities of software. Xu et al. [30] introduce a holistic Chinese benchmark named SuperCLUE for evaluating the effectiveness of LLMs. Hu et al. [60] have embraced a hybrid approach, merging both qualitative and quantitative methodologies, predicated on the Fuzzy Comprehensive Evaluation (FCE) framework.

8. Conclusion

This paper presents a novel validation approach for indicator systems using Metamorphic Testing (MT), addressing the challenge of the Test Oracle problem in scenarios lacking explicit expected outputs. By crafting system-specific metamorphic relations and employing a dynamic test case generation strategy, the method has proven effective in validating real-world systems. Our empirical studies, including mutation testing with an average mutation score of 0.83, surpass traditional methods with a score of 0.65, highlighting MT's superior error detection capabilities. The proposed nine metamorphic relations, tailored to indicator systems, demonstrate practical value and adaptability across various applications, such as red wine quality and ecological-economic benefit evaluations. However, the current relations do not fully cover all error patterns, indicating a need for expansion with domain knowledge. Additionally, the efficiency of test case generation could be enhanced through reinforcement learning. Future work will focus on creating an automated tool chain for large-scale system validation and integrating MT with formal validation methods to improve validation completeness.

CRediT authorship contribution statement

GuoHao Ma: Software, Visualization, Writing – original draft preparation. **Bo Yang:** Conceptualization, Methodology, Formal analysis, Writing – review & editing. **XiaoKai Xia:** Methodology, Review. **Luo Xu:** Investigation, Methodology.

Declaration of competing interest

The authors declare no conflicts of interest.

Acknowledgment

This research was funded by National Key R&D Program of China, grant number 2022YFF1302700, the Outstanding Youth Team of Central Universities, grant number QNTD202504.

Data availability

The data that support the findings of this study are openly available at https://pan.baidu.com/s/1rjgDKreFjLyA9FckObgmyQ?pwd=32bi, extraction code 32bi.

References

[1] Robert M. O'Keefe, Daniel E. O'Leary, Expert system verification and validation: a survey and tutorial, Artif. Intell. Rev. 7 (1993) 3–42.
[2] Richard J. Shavelson, Lorraine McDonnell, Jeannie Oakes, What are educational indicators and indicator systems? Pr. Assess. Res. Eval. 2 (1) (1990).
[3] Chiyang Gao, Wenbing Luo, Jian Wang, Yingying Zhang, Software quality evaluation model based on multiple linear regression and fuzzy comprehensive evaluation method, in: 2022 9th International Conference on Dependable Systems and their Applications, DSA, IEEE, 2022, pp. 383–389.
[4] Yuhang Zhao, Ruigang Liang, Xiang Chen, Jing Zou, Evaluation indicators for open-source software: a review, Cybersecurity 4 (2021) 1–24.
[5] Tianyun Zhang, Jun Zhang, Feiyue Wang, Peidong Xu, Tianlu Gao, Haoran Zhang, Ruiqi Si, Parallel system based quantitative assessment and self-evolution for artificial intelligence of active power corrective control, CSEE J. Power Energy Syst. 10 (1) (2023) 13–28.
[6] Sangwon Hyun, Mingyu Guo, M. Ali Babar, METAL: Metamorphic testing framework for analyzing large-language model qualities, in: 2024 IEEE Conference on Software Testing, Verification and Validation, ICST, IEEE, 2024, pp. 117–128.
[7] Felipe Febrero, M. Angeles Moraga, Coral Calero, Software reliability as user perception: application of the fuzzy analytic hierarchy process to software reliability analysis, in: 2017 IEEE International Conference on Software Quality, Reliability and Security, QRS, IEEE, 2017, pp. 224–231.
[8] Qian Zheng, Shui-Long Shen, Annan Zhou, Hai-Min Lyu, Inundation risk assessment based on G-DEMATEL-AHP and its application to zhengzhou flooding disaster, Sustain. Cities Soc. 86 (2022) 104138.
[9] Xaimarie Hernández-Cruz, Jesus R. Villalobos, George Runger, Grace Neal, Building an intelligent system to identify trends in agricultural markets, J. Clean. Prod. 425 (2023) 138956.
[10] Carla Tognato De Oliveira, Thales Eduardo Tavares Dantas, Sebastiao Roberto Soares, Nano and micro level circular economy indicators: Assisting decision-makers in circularity assessments, Sustain. Prod. Consum. 26 (2021) 455–468.
[11] Wenge Qiu, Yang Liu, Feng Lu, Guang Huang, Establishing a sustainable evaluation indicator system for railway tunnel in China, J. Clean. Prod. 268 (2020) 122150.
[12] Ali Ebadi Torkayesh, Dragan Pamucar, Fatih Ecer, Prasenjit Chatterjee, An integrated BWM-LBWA-cocoso framework for evaluation of healthcare sectors in eastern europe, Socio-Economic Plan. Sci. 78 (2021) 101052.

[13] Pingfang Yang, Research on the evaluation indicator system, in: Proceedings of the 2nd International Conference on Humanities, Wisdom Education and Service Management, HWESM 2023, vol. 760, Springer Nature, 2023, p. 303.

[14] Earl T Barr, Mark Harman, Phil McMinn, Muzammil Shahbaz, Shin Yoo, The oracle problem in software testing: A survey, IEEE Trans. Softw. Eng. 41 (5) (2014) 507–525.

[15] Shreya Shankar, Labib Fawaz, Karl Gyllstrom, Aditya Parameswaran, Automatic and precise data validation for machine learning, in: Proceedings of the 32nd ACM International Conference on Information and Knowledge Management, 2023, pp. 2198–2207.

[16] Houssem Ben Braiek, Foutse Khomh, On testing machine learning programs, J. Syst. Softw. 164 (2020) 110542.

[17] Tsong Yueh Chen, Fei-Ching Kuo, Huai Liu, Pak-Lok Poon, Dave Towey, TH Tse, Zhi Quan Zhou, Metamorphic testing: A review of challenges and opportunities, ACM Comput. Surv. 51 (1) (2018) 1–27.

[18] Xiaoyuan Xie, Joshua WK Ho, Christian Murphy, Gail Kaiser, Baowen Xu, Tsong Yueh Chen, Testing and validating machine learning classifiers by metamorphic testing, J. Syst. Softw. 84 (4) (2011) 544–558.

[19] Rui Li, Huai Liu, Pak-Lok Poon, Dave Towey, Chang-Ai Sun, Zheng Zheng, Zhi Quan Zhou, Tsong Yueh Chen, Metamorphic relation generation: State of the art and research directions, ACM Trans. Softw. Eng. Methodol. 34 (5) (2025) 1–25.

[20] Yao Deng, Xi Zheng, Tianyi Zhang, Huai Liu, Guannan Lou, Miryung Kim, Tsong Yueh Chen, A declarative metamorphic testing framework for autonomous driving, IEEE Trans. Softw. Eng. 49 (4) (2022) 1964–1982.

[21] Tsong Y. Chen, Shing C. Cheung, Shiu Ming Yiu, Metamorphic testing: a new approach for generating next test cases, 2020, arXiv preprint arXiv:2002.12543.

[22] Yue Jia, Mark Harman, An analysis and survey of the development of mutation testing, IEEE Trans. Softw. Eng. 37 (5) (2010) 649–678.

[23] Bo Yang, Ji Wu, Chao Liu, Regression test case generation based on variable impact analysis and data mutation, J. Comput. Sci. Tech. 39 (11) (2016) 2372–2387.

[24] Congying Xu, Songqiang Chen, Jiarong Wu, Shing-Chi Cheung, Valerio Terragni, Hengcheng Zhu, Jialun Cao, MR-adopt: Automatic deduction of input transformation function for metamorphic testing, in: Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering, 2024, pp. 557–569.

[25] Jingxing Wang, Haitao Wang, Ying Jiang, Xing Chen, Research on software quality evaluation model based on triangular fuzzy number analytic hierarchy process, Comput. Digit. Eng. 45 (9) (2017) 1693.

[26] Paulo Cortez, António Cerdeira, Fernando Almeida, Telmo Matos, José Reis, Modeling wine preferences by data mining from physicochemical properties, Decis. Support Syst. 47 (4) (2009) 547–553.

[27] Li-yan Sun, Cheng-lin Miao, Li Yang, Ecological-economic efficiency evaluation of green technology innovation in strategic emerging industries based on entropy weighted TOPSIS method, Ecol. Indic. 73 (2017) 554–558.

[28] Joe W. Duran, Simeon C. Ntafos, An evaluation of random testing, IEEE Trans. Softw. Eng. SE-10 (4) (1984) 438–444.

[29] Yu-Seung Ma, Jeff Offutt, Yong Rae Kwon, MuJava: an automated class mutation system, Softw. Test. Verif. Reliab. 15 (2) (2005) 97–133.

[30] Liang Xu, Anqi Li, Lei Zhu, Hang Xue, Changtai Zhu, Kangkang Zhao, Haonan He, Xuanwei Zhang, Qiyue Kang, Zhenzhong Lan, Superclue: A comprehensive chinese large language model benchmark, 2023, arXiv preprint arXiv:2307.15020.

[31] Chang ai Sun, Baoli Liu, An Fu, Yiqiang Liu, Huai Liu, Path-directed source test case generation and prioritization in metamorphic testing, J. Syst. Softw. 183 (2022) 111091.

[32] Congying Xu, Songqiang Chen, Jiarong Wu, Shing-Chi Cheung, Valerio Terragni, Hengcheng Zhu, Jialun Cao, MR-adopt: Automatic deduction of input transformation function for metamorphic testing, in: Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering, ASE '24, Association for Computing Machinery, New York, NY, USA, 2024, pp. 557–569.

[33] Congying Xu, Valerio Terragni, Hengcheng Zhu, Jiarong Wu, Shing-Chi Cheung, MR-scout: Automated synthesis of metamorphic relations from existing test cases, ACM Trans. Softw. Eng. Methodol. 33 (6) (2024).

[34] Zhihao Ying, Dave Towey, Anthony Graham Bellotti, Tsong Yueh Chen, Zhi Quan Zhou, SFIDMT-ART: A metamorphic group generation method based on adaptive random testing applied to source and follow-up input domains, Inf. Softw. Technol. 175 (2024) 107528.

[35] Mingyue Jiang, Tsong Yueh Chen, Shuai Wang, On the effectiveness of testing sentiment analysis systems with metamorphic testing, Inf. Softw. Technol. 150 (2022) 106966.

[36] Xiaoyuan Xie, Zhiyi Zhang, Tsong Yueh Chen, Yang Liu, Pak-Lok Poon, Baowen Xu, METTLE: A metamorphic testing approach to assessing and validating unsupervised machine learning systems, IEEE Trans. Reliab. 69 (4) (2020) 1293–1322.

[37] Nazanin Bayati Chaleshtari, Fabrizio Pastore, Arda Goknil, Lionel C Briand, Metamorphic testing for web system security, IEEE Trans. Softw. Eng. 49 (6) (2023) 3430–3471.

[38] Zhi Quan Zhou, Shaowen Xiang, Tsong Yueh Chen, Metamorphic testing for software quality assessment: A study of search engines, IEEE Trans. Softw. Eng. 42 (3) (2015) 264–284.

[39] Chang-Ai Sun, Hepeng Dai, Huai Liu, Tsong Yueh Chen, Feedback-directed metamorphic testing, ACM Trans. Softw. Eng. Methodol. 32 (1) (2023) 1–34.

[40] Quang-Hung Luu, Man F Lau, Sebastian PH Ng, Tsong Yueh Chen, Testing multiple linear regression systems with metamorphic testing, J. Syst. Softw. 182 (2021) 111062.

[41] Dongwei Xiao, Zhibo Liu, Yuanyuan Yuan, Qi Pang, Shuai Wang, Metamorphic testing of deep learning compilers, Proc. ACM Meas. Anal. Computing Syst. 6 (1) (2022) 1–28.

[42] Quang-Hung Luu, Huai Liu, Tsong Yueh Chen, Hai L. Vu, A sequential metamorphic testing framework for understanding autonomous vehicle's decisions, IEEE Trans. Intell. Veh. (2024).

[43] Matteo Paltenghi, Michael Pradel, Morphq: Metamorphic testing of the qiskit quantum computing platform, in: 2023 IEEE/ACM 45th International Conference on Software Engineering, ICSE, IEEE, 2023, pp. 2413–2424.

[44] Jon Ayerdi, Pablo Valle, Sergio Segura, Aitor Arrieta, Goiuria Sagardui, Maite Arratibel, Performance-driven metamorphic testing of cyber-physical systems, IEEE Trans. Reliab. 72 (2) (2022) 827–845.

[45] Rodolfo Adamshuk Silva, Simone do Rocio Senger de Souza, Paulo Sérgio Lopes de Souza, A systematic review on search based mutation testing, Inf. Softw. Technol. 81 (2017) 19–35.

[46] Goran Petrović, Marko Ivanković, Gordon Fraser, René Just, Does mutation testing improve testing practices? in: 2021 IEEE/ACM 43rd International Conference on Software Engineering, ICSE, IEEE, 2021, pp. 910–921.

[47] Goran Petrović, Marko Ivanković, Gordon Fraser, René Just, Practical mutation testing at scale: A view from google, IEEE Trans. Softw. Eng. 48 (10) (2021) 3900–3912.

[48] Samuel J Kaufman, Ryan Featherman, Justin Alvin, Bob Kurtz, Paul Ammann, René Just, Prioritizing mutants to guide mutation testing, in: Proceedings of the 44th International Conference on Software Engineering, 2022, pp. 1743–1754.

[49] Arghavan Moradi Dakhel, Amin Nikanjam, Vahid Majdinasab, Foutse Khomh, Michel C Desmarais, Effective test generation using pre-trained large language models and mutation testing, Inf. Softw. Technol. 171 (2024) 107468.

[50] Renzo Degiovanni, Mike Papadakis, μBert: Mutation testing using pre-trained language models, in: 2022 IEEE International Conference on Software Testing, Verification and Validation Workshops, ICSTW, IEEE, 2022, pp. 160–169.

[51] Ana B Sánchez, Pedro Delgado-Pérez, Inmaculada Medina-Bulo, Sergio Segura, Mutation testing in the wild: findings from GitHub, Empir. Softw. Eng. 27 (6) (2022) 132.

[52] Daniel Fortunato, José Campos, Rui Abreu, Mutation testing of quantum programs: A case study with Qiskit, IEEE Trans. Quantum Eng. 3 (2022) 1–17.

[53] Jinlei Sun, Song Huang, Changyou Zheng, Tingyong Wang, Cheng Zong, Zhanwei Hui, Mutation testing for integer overflow in ethereum smart contracts, Tsinghua Sci. Technol. 27 (1) (2021) 27–40.

[54] Marcio Eduardo Delamaro, J.C. Maidonado, Aditya P. Mathur, Interface mutation: An approach for integration testing, IEEE Trans. Softw. Eng. 27 (3) (2001) 228–247.

[55] Moises Rodriguez, Mario Piattini, Christof Ebert, Software verification and validation technologies and tools, IEEE Softw. 36 (2) (2019) 13–24.

[56] Victor R. Basili, Lionel C. Briand, Walcélio L. Melo, A validation of object-oriented design metrics as quality indicators, IEEE Trans. Softw. Eng. 22 (10) (1996) 751–761.

[57] Tibor Gyimóthy, Rudolf Ferenc, Istvan Siket, Empirical validation of object-oriented metrics on open source software for fault prediction, IEEE Trans. Softw. Eng. 31 (10) (2005) 897–910.

[58] Soohyun Park, Hankyul Baek, Jung Won Yoon, Youn Kyu Lee, Joongheon Kim, AQUA: Analytics-driven quantum neural network (QNN) user assistance for software validation, Future Gener. Comput. Syst. 159 (2024) 545–556.

[59] Quentin Rouland, Brahim Hamid, Jason Jaskolka, A model-driven formal methods approach to software architectural security vulnerabilities specification and verification, J. Syst. Softw. 219 (2025) 112219.

[60] Yanjuan Hu, Lizhe Wu, Xueqiao Pan, Zhanli Wang, Xiaoxia Xu, Comprehensive evaluation of cloud manufacturing service based on fuzzy theory, Int. J. Fuzzy Syst. 23 (6) (2021) 1755–1764.