Low-Cost Functional Testing Based on Data Imputation Integrating Fault Tree Analysis and XGBoost

Yiyang Xu, Kangcheng Wang, Yun-Bo Zhao, *Senior Member, IEEE*, Yu Kang, *Senior Member, IEEE*, and Peng Bai

***Abstract*—Functional testing is essential for ensuring the quality of electronic products. As system complexity increases, the cost of functional testing—particularly during the motherboard testing stage—has risen significantly. Designing an efficient testing strategy is therefore key to reducing overall testing costs. Although data imputation methods can enhance the effectiveness of strategies by addressing missing data, current approaches do not adequately account for the impact of correlation information between system modules on the input information of the data imputation model. This oversight results in suboptimal imputation performance, making it challenging to further reduce testing costs. To overcome this limitation, we propose a data imputation method that integrates fault tree analysis (FTA) with eXtreme gradient boosting (XGBoost), effectively combining system fault mode analysis with data-driven modeling. First, faults in nonbus data transmission are additionally considered, and an enhanced fault tree for motherboard testing is constructed. Next, correlations among system modules are quantitatively represented as system event associations within the fault tree, based on which the attributes requiring imputation are determined. Then, a top–down quantitative analysis method driven by both mechanisms and data is introduced to infer the states of basic events from intermediate event states. This mapping from intermediate events to basic events reduces the proportion of missing data. Based on this, missing values are imputed using the tree-based XGBoost model. Experiments conducted on transformed real-world manufacturing data demonstrate that effective testing strategies can be dynamically developed under varying conditions using the proposed method. Testing costs are reduced by up to 5.64% in high-yield scenarios, 6.66% in low-yield scenarios, and 7.07% during long-term yield fluctuations. Furthermore, the defect level is reduced by as much as 51.02%.**

Received 10 April 2025; revised 29 May 2025; accepted 7 July 2025. Date of publication 15 July 2025; date of current version 22 August 2025. This work was supported by the Key Research and Development Program of Anhui under Grant 24KT005. Recommended for publication by Associate Editor A. Watanabe upon evaluation of reviewers' comments. *(Corresponding author: Kangcheng Wang.)*

Yiyang Xu and Kangcheng Wang are with the AHU-IAI AI Joint Laboratory, Anhui University, Hefei 230601, China, and also with the Institute of Artificial Intelligence, Hefei Comprehensive National Science Center, Hefei 230088, China (e-mail: kcwang@ustc.edu.cn).

Yun-Bo Zhao and Yu Kang are with the AHU-IAI AI Joint Laboratory, Anhui University, Hefei 230601, China, also with the Institute of Artificial Intelligence, Hefei Comprehensive National Science Center, Hefei 230088, China, and also with the Department of Automation and the Institute of Advanced Technology, University of Science and Technology of China, Hefei 230026, China.

Peng Bai is with the Department of Automation, University of Science and Technology of China, Hefei 230026, China.

Digital Object Identifier 10.1109/TCPMT.2025.3589029

***Index Terms*—Data imputation, eXtreme gradient boosting (XGBoost), fault tree analysis (FTA), functional testing, intelligent manufacturing.**

I. Introduction

THE functional test is one of the important processes to ensure the functionality of circuit boards [1]. With the booming electronics industry, the complexity of circuit boards in modern electronic devices has been increasing rapidly [2]. Consequently, the cost of functional testing has become a significant portion of the overall manufacturing cost [3]. Therefore, reducing the cost of testing through intelligent manufacturing methods has proven to be an effective strategy for increasing the profit margin of semiconductor manufacturers and has gained significant attention in recent years [4], [5].

For electronic products equipped with motherboards, the functional test typically consists of two stages: the motherboard functional test and the finished product functional test [6]. Both stages test multiple functional items to identify any defective boards [7]. The essential functions of the board will be tested in both types of functional tests, while additional functions will be tested in the finished product functional test [8]. The testing strategy at the motherboard stage determines which test items will be tested, usually represented as a binary logic vector [9]. In order to effectively balance the testing time cost and the repair cost of unidentified defective products, the testing strategy typically only tests a portion of the test items [10]. This results in missing data for the untested items, and their true values can only be obtained during the finished product testing stage several hours later [11]. This seriously impacts the real-time adjustment of motherboard testing strategies. Given the variable operating conditions in actual production, a significant delay in the testing strategy can lead to high testing costs [12]. By utilizing data imputation methods to fill in missing data online, it is feasible to adapt the testing strategy promptly in response to variations in operating conditions, thus efficiently reducing testing costs.

Existing data imputation methods can be categorized into three groups: methods relying on simple statistical analysis, methods utilizing traditional machine learning models, and methods employing deep learning algorithms [13]. Among them, methods based on simple statistical analysis have low model complexity, making it difficult to effectively impute data

for complex manufacturing processes [14]. Methods based on deep learning algorithms can achieve good results in fields, such as image processing, but due to the long training time, it is difficult to ensure the real-time nature of data imputation [15]. Additionally, these methods have high uncertainty in results, challenging reliability assurance in industrial manufacturing scenarios with stringent reliability requirements [16]. On the other hand, methods based on traditional machine learning have advantages, such as a simple model structure, short model training time, and strong robustness [17]. These methods can provide stable data imputation results online, making them the preferred choice for manufacturing data imputation [18].

Among the data imputation methods based on traditional machine learning models, using decision tree models is a common approach [19]. The tree-like structure of decision tree models provides strong interpretability, and their capability to effectively manage missing values in discrete variables facilitates widespread data imputation across various datasets [20], [21]. In recent years, various types of decision tree-based methods have been studied for data imputation. These methods include the standard decision tree [22], random forests [23], and tree-based eXtreme gradient boosting (XGBoost) [24]. Recent studies have shown that tree-based XGBoost is often preferred over other types of decision tree models due to its higher accuracy and effective control of model complexity [25], [26], [27]. However, the existing data imputation methods that rely on data-driven modeling often fail to fully account for the influence of the correlation information between system modules on the input information of the data imputation model. This oversight leads to suboptimal imputation performance and complicates efforts to further reduce testing costs. If the system fault mode analysis technology can accurately quantify the correlations between modules and integrate this information into the data imputation model, it is expected to enhance the accuracy of data imputation [28], [29].

Fault tree analysis (FTA) is a representative system failure mode analysis technique [30]. Compared to similar methods, FTA can analyze system fault information through both qualitative and quantitative analyses simultaneously [31]. Therefore, it has been widely studied and applied in various fields, such as the aircraft industry [32] and power systems [33]. FTA has also been explored in manufacturing processes for fault analysis. By focusing on analyzing faults caused by environmental and human factors, effective fault information from manufacturing processes has been obtained [34], [35]. Considering the high level of automation in electronic manufacturing production lines nowadays, fewer faults are caused by human factors, while most of the faults are caused by manufacturing procedures [36]. Therefore, in our preliminary work, we focused on analyzing defects caused by manufacturing procedures, established a fault tree for the motherboard functional testing process, and preliminarily designed a testing strategy for motherboards based on the analysis results [37]. However, the fault tree in the aforementioned study does not cover a comprehensive range of defect types corresponding to the events. This limitation complicates the effective quantification of the correlations between the attributes that require imputation and other attributes, necessitating further in-depth research.

Based on the above analysis, this article integrates system fault mode analysis with data-driven modeling techniques and proposes a data imputation method based on FTA and XGBoost. First, faults in nonbus data transmission are additionally considered, and an improved fault tree for the motherboard functional testing process is established. Then, by quantifying the correlations among system modules as shared basic events and specific basic events within the fault tree, the attributes requiring data imputation are determined based on both shared and specific basic events present in these system events. Furthermore, a top–down fault tree quantitative analysis method is developed by integrating data and mechanisms to infer basic event states from intermediate event states, which maps the focus of data imputation from intermediate events to basic events. Finally, the tree-based XGBoost model is employed to perform data imputation. Effective testing strategies can be dynamically obtained under varying conditions with the proposed method. The methods outlined in this article are anticipated to facilitate the cost-effective and intelligent advancement of the electronics manufacturing industry.

The arrangement of the remaining parts of this article is as follows: Section II provides an overview of the functional testing process and introduces the main theories involved in the proposed methods; Section III establishes a fault tree for the functional testing process of the motherboard, determines the attributes to be imputed based on this, constructs a top–down fault tree quantitative analysis method, uses a tree-based XGBoost model for data imputation, and lists the method evaluation criteria; Section IV verifies the effectiveness of the proposed methods through experiments; finally, a summary of the entire article is provided.

II. Background

A. Introduction to Functional Testing

Functional testing is one of the most crucial tests to verify the functionality of circuit boards. The laptop functional test process is shown in Fig. 1, which can be divided into two main stages: the motherboard functional test and the finished product functional test. The motherboard functional test mainly assesses the essential functions of the board. In the motherboard functional test, items will be tested sequentially. If a faulty item is identified during the testing, the motherboard will be sent for repair. If all items are tested and none are faulty, the motherboard will then be assembled and will undergo the finished product functional test. The finished product functional test will test more functions in addition to identifying more potential defects on the motherboard. The defective product will be sent for repair if a faulty item is identified in the finished product functional test. Products that pass the finished product functional test will be transferred to the next stage of the production line.

The repair cost of a defective motherboard is lower than the repair cost of a defective product. Therefore, designing an efficient test strategy for the motherboard functional test is essential for balancing the testing time cost and the repair costs.

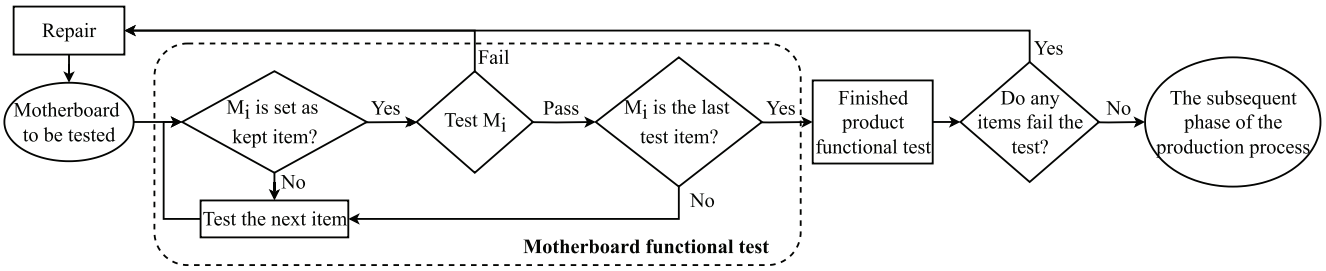

Fig. 1. Flowchart of functional testing.

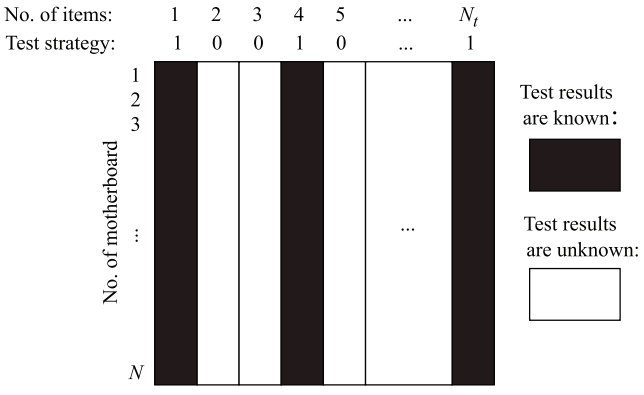

Fig. 2. Results of motherboard functional testing with missing data.

Most functional test strategies rely on attribute selection, where only specific items are tested per motherboard, while others are skipped. If all tested items pass but untested items contain faults, the defective motherboard remains unidentified, leading to missing identified defective motherboards. In other words, a testing strategy that relies on attribute selection may lead to missing data because of the presence of untested items.

In a laptop motherboard functional test, the test strategy and results are shown in Fig. 2. For N_t test items and N boards, the "1/0" in the test strategy denotes "to be tested" and "untested," respectively. The items to be tested are kept items, while the untested items are considered redundant items. If a method can accurately impute untested results and identify faults within untested items, the results of the motherboard functional test can be significantly enhanced. Data imputation is a method used to address the missing data arising from the test strategy.

B. Basic Theories

1) Fault Tree Analysis: The fault tree is a logic diagram that describes the causal relationships between different types of faults and their underlying causes. FTA utilizes fault trees for fault diagnosis and reliability analysis. The construction of a fault tree follows a top–down approach. Initially, the research object is analyzed to categorize system events into intermediate events, basic events, and the top event.

The top event represents the system-level fault that system administrators hope to avoid. In the motherboard functional test, the top event is the motherboard being identified as defective. By tracing back the top event and investigating its possible causes, intermediate events can be identified, related to module faults. Suppose the causes of intermediate events are traced back until the causes of system events cannot be further divided. In that case, these system events are considered basic events, serving as the root causes of the top event.

Intermediate events typically consist of the logical connection between basic events or other intermediate events, with the logical connection represented as logic gates in the fault tree. The logical connections primarily involve OR and AND, corresponding to the logic gates: OR gate and AND gate, respectively. After the fault tree is established, the causal relationships between system events can be quantified into Boolean expressions based on the fault tree, thereby quantifying the causal associations between system events using Boolean expressions. The states of the top event and intermediate events can be translated into the logical operation results of basic events.

2) eXtreme Gradient Boosting: XGBoost is an ensemble learning algorithm that trains multiple base learners and integrates their results into the output. XGBoost is an enhanced version of gradient boosting decision trees (GBDTs). Each iteration in GBDT aims to fit the residual transferred from the previous iteration, a principle also followed by XGBoost. Taylor's formula is employed to compute the first-order approximation of the loss function in GBDT. XGBoost, on the other hand, uses Taylor's formula to calculate a second-order approximation of the loss function. This method is applicable to various loss functions and leads to improved accuracy. Moreover, XGBoost offers support for regularization and aligned sampling.

III. Method

In this section, the fault tree of the motherboard functional test is established, and the imputed attributes are then determined by FTA and stratified sampling. Following the imputed attribute mapping method based on FTA, the missing attributes are imputed using XGBoost. Additionally, the evaluation criteria are provided.

A. Fault Tree Establishment

During the motherboard functional test, the mechanism information of the motherboard can be analyzed to establish the motherboard functional test fault tree, which can help design the test strategy by improving the test strategy with hardware connections and other intrinsic information. Set the

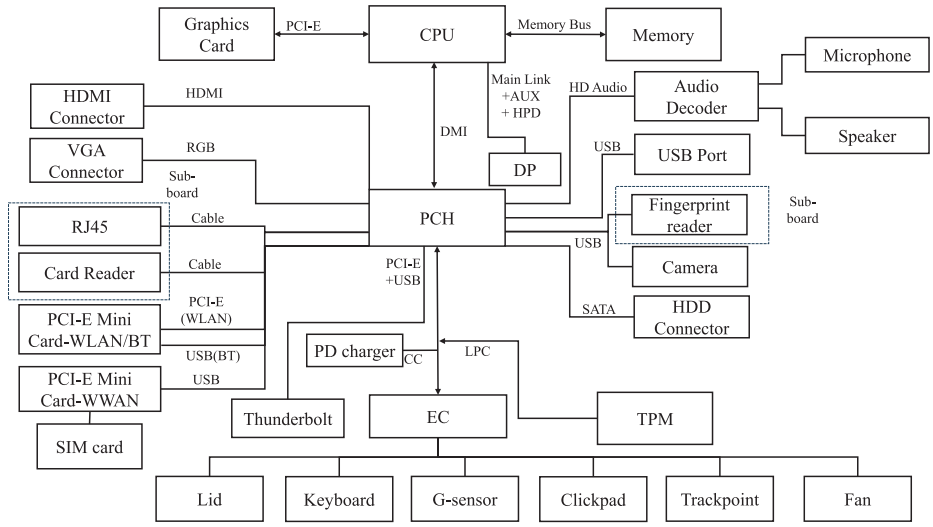

Fig. 3. Simplified block diagram of a laptop motherboard.

TABLE I
TEST ITEMS OF A LAPTOP MOTHERBOARD

No.	Test item	No.	Test item
1	HDMI test	15	Keyboard test
2	DP test	16	TPM test
3	VGA test	17	USB test 1
4	Card reader test	18	USB test 2
5	WLAN test	19	USB TYPE-C test
6	Bluetooth test	20	Fingerprint reader test
7	WWAN test	21	Camera test
8	SIM card test	22	Network port test
9	SATA test	23	Microphone test
10	Trackpoint test	24	Speaker test
11	Fan test	25	PD charger test
12	Lid test	26	Thunderbolt test
13	G-sensor test	27	Memory test
14	Clickpad test	28	Graphics card test

tested motherboard identified as defective as the top event. Set the functional module faults as the intermediate event. Undivided faults, such as component faults and bus faults, are set as basic events.

As shown in Fig. 3, the HDMI and VGA connectors are linked to the Platform Controller Hub (PCH) module through corresponding buses. The RJ45 and card reader are linked to the PCI-E bus through a local connection cable. WLAN, Bluetooth, and WWAN on the PCI-E mini card are linked to the PCH module through their corresponding buses. The lid, keyboard, clickpad, and so on are linked to the EC module via a local connection cable and are further connected to the PCH module. The microphone and speaker are linked to the audio decoder through corresponding local connection cables and are further connected to the PCH module via the HD Audio bus. The fingerprint reader is located on a subboard linked to the USB bus through a local connection cable. The USB bus links the camera and USB port to the PCH module. The PCH module interacts with the central processing unit (CPU) module through the DMI bus. This laptop motherboard contains 28 essential functional modules, corresponding to 28 test items, as is shown in Table. 1. Any test item identified as faulty will lead to the corresponding motherboard being identified as defective. Any subevents of a test item's corresponding intermediate event identified as "fail" will lead to the test item being identified as faulty. Based on the analysis mentioned above, all logic gates in the motherboard functional test fault tree should be OR gates.

The laptop motherboard can be divided into four parts based on its functional modules: the embedded controller module (EC), the PCH, the CPU, and the essential functional module. The EC module consists of the EC BIOS and the EC chip. Therefore, faults in the EC module can be categorized into two scenarios: EC BIOS fault and EC chip fault.

Due to production process issues, chip faults can be further classified into two situations. The first issue is when the components on the chip fail, which can be attributed to component damage. The second issue might occur when the component is soldered onto the motherboard; the oxidation of solder paste or component termination may cause the pseudo-soldering defect. This defect would weaken interconnections, making them prone to failure under thermal cycling or vibration. Therefore, the EC chip fault can be categorized into two types: EC chip damage and EC pseudo soldering.

The CPU module exclusively contains the CPU chip, while the PCH module solely incorporates the PCH chip. Consequently, faults in these chips should be classified into two categories analogous to EC chip failures: chip damage and pseudo soldering. However, critical functional dependencies exist: the EC module must first communicate with the PCH module during operation, which subsequently interfaces with the CPU module. This hierarchical connectivity implies that EC module faults may originate from cascading failures in either the PCH or CPU modules, while the PCH module fault often correlates with the underlying CPU module fault.

Four primary fault modalities are observed in essential functional modules: fault propagation from associated CPU, PCH, or EC modules; component faults; bus communication failures; and connection faults. Connection faults are data transmission faults on nonbus data transmissions, which always co-occur

TABLE II
INTERMEDIATE EVENTS OF THE LAPTOP MOTHERBOARD'S FAULT TREE

Symbol	Fault	Symbol	Fault
M_1	HDMI fault	M_{17}	USB-1 fault
M_2	DP fault	M_{18}	USB-2 fault
M_3	VGA fault	M_{19}	USB TYPE-C fault
M_4	Card reader fault	M_{20}	Fingerprint reader fault
M_5	WLAN fault	M_{21}	Camera fault
M_6	Bluetooth fault	M_{22}	Network port fault
M_7	WWAN fault	M_{23}	Microphone fault
M_8	SIM card fault	M_{24}	Speaker fault
M_9	SATA fault	M_{25}	PD charger fault
M_{10}	Trackpoint fault	M_{26}	Thunderbolt fault
M_{11}	Fan fault	M_{27}	Memory fault
M_{12}	Lid fault	M_{28}	Graphics card fault
M_{13}	G-sensor fault	M_{29}	CPU module fault
M_{14}	Clickpad fault	M_{30}	EC module fault
M_{15}	Keyboard fault	M_{31}	PCH module fault
M_{16}	TPM fault		

with component faults and bus failures, making them easily misclassified.

The card reader resides on a PCI-E subboard, making it susceptible to connection faults. When the PCH module broadcasts queries, three failure modes may prevent the response. The first possibility is that the card reader is defective, preventing it from accepting queries sent from the PCH or responding to the PCH. The second possibility is that a bus fault or other bus-related defects cause the bus's data transmission to fail. The third possibility is a connection fault. The card reader responds while the response is blocked by the connection fault between the subboard and the bus.

The fault tree for the motherboard functional test, as depicted in Fig. 4, has been developed based on the analysis mentioned earlier. The associated intermediate events and basic events can be found in Tables II and III.

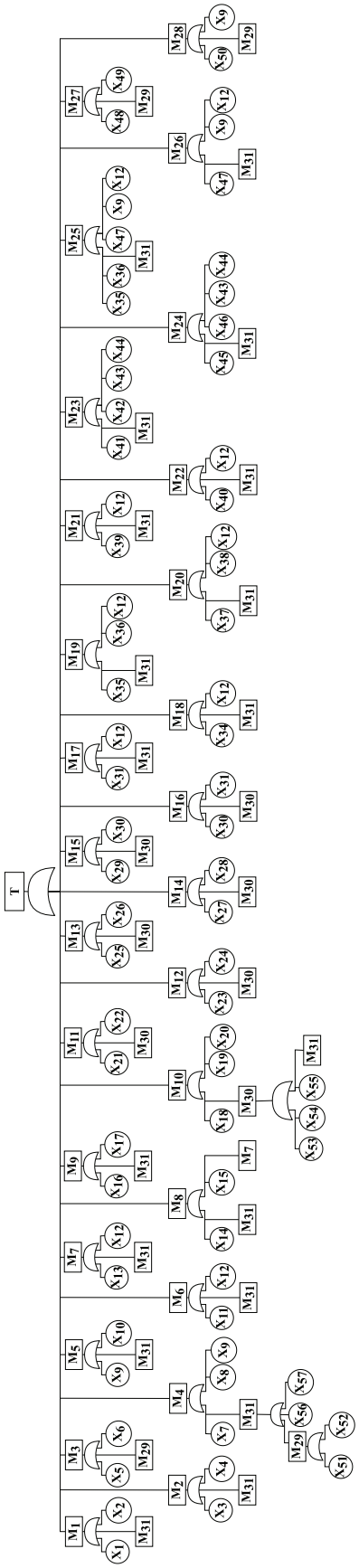

Fig. 4. Fault tree of a laptop motherboard.

B. Imputed Attribute Determination by FTA and Stratified Sampling

This section proposes a two-round attribute selection method to determine attributes requiring imputation. The first-round attribute division employs FTA to prioritize attributes containing the most specific basic events. The second-round attribute selection implements stratified sampling based on the distribution of shared basic events to choose from the remaining set, thereby incorporating supplementary basic event information.

1) Attribute Division Based on FTA: This section proposes an attribute division method based on FTA to classify attributes, prioritizing those containing the most specific basic events. The second-round attribute selection similarly leverages the attribute division results to preferentially select attributes with richer specific basic events.

In the motherboard functional testing, the intermediate event state corresponding to each attribute can be translated into a Boolean expression composed solely of basic events. Multiple Boolean expressions may contain the same basic events, which are defined as shared basic events in this article. Except for these basic events, other basic events appear in particular Boolean expressions. The latter type, whose state can only

TABLE III
BASIC EVENTS OF THE LAPTOP MOTHERBOARD'S FAULT TREE

Symbol	Fault	Symbol	Fault	Symbol	Fault
X_1	HDMI component fault	X_{20}	Trackpoint-EC connection fault	X_{39}	Camera component fault
X_2	HDMI bus fault	X_{21}	Fan component fault	X_{40}	Network port component fault
X_3	VGA component fault	X_{22}	Fan connection fault	X_{41}	Microphone component fault
X_4	RGB bus fault	X_{23}	Lid component fault	X_{42}	Microphone connection fault
X_5	DP component fault	X_{24}	Lid connection fault	X_{43}	HD-audio bus fault
X_6	AUX bus fault	X_{25}	G-sensor component fault	X_{44}	Audio decoder fault
X_7	Card reader component fault	X_{26}	G-sensor connection fault	X_{45}	Speaker component fault
X_8	Card reader connection fault	X_{27}	Clickpad component fault	X_{46}	Speaker connection fault
X_9	PCI-E bus fault	X_{28}	Clickpad connection fault	X_{47}	Thunderbolt component fault
X_{10}	WLAN component fault	X_{29}	Keyboard component fault	X_{48}	Memory component fault
X_{11}	Bluetooth component fault	X_{30}	Keyboard connection fault	X_{49}	Memory bus fault
X_{12}	USB bus fault	X_{31}	TPM component fault	X_{50}	Graphics card component fault
X_{13}	WWAN component fault	X_{32}	LPC bus fault	X_{51}	CPU chip damage
X_{14}	SIM card component fault	X_{33}	USB-1 component fault	X_{52}	CPU pseudo soldering
X_{15}	SIM card connection fault	X_{34}	USB-2 component fault	X_{53}	EC chip damage
X_{16}	SATA component fault	X_{35}	USB TYPE-C component fault	X_{54}	EC pseudo soldering
X_{17}	SATA bus fault	X_{36}	PD charger component fault	X_{55}	EC BIOS fault
X_{18}	Trackpoint component fault	X_{37}	Fingerprint reader component fault	X_{56}	PCH pseudo soldering
X_{19}	Trackpoint-Keyboard connection fault	X_{38}	Fingerprint reader connection fault	X_{57}	PCH chip damage

be inferred from one specific intermediate event, is defined as a specific basic event in this article.

The specific basic events reflect the unique structural or functional characteristics of their corresponding attributes. In contrast, shared basic events, which appear in multiple Boolean expressions, represent shared correlation information among different system modules. Since the state of a shared basic event can be inferred through any of the intermediate events in which it appears, its information tends to be considered redundant or non-valuable. Conversely, the state inference of a specific basic event depends solely on its corresponding attribute, which must be identified first to enable accurate inference. As the state of a specific basic event cannot be inferred from any other attribute, it consistently provides valuable basic event information. During attribute selection, it is crucial to prioritize attributes containing more specific basic events in order to reduce redundancy and enhance the effectiveness of the data imputation process.

The above analysis concludes that compared to other intermediate events, those containing more specific basic events provide more valuable basic event information for basic event state inference and should therefore be prioritized.

Fig. 5 shows four fault branches of the laptop motherboard's fault tree, namely, (a) the microphone fault branch, (b) the speaker fault branch, (c) the PCH module fault branch, and 5(d) the CPU fault branch. As shown in Fig. 5(a) and (b), it can be observed that when analyzing M_{23} and M_{24} individually, each includes four specific basic events. However, X_{43} and X_{44} are present in both sets of basic events, indicating that M_{24} will only contain two specific basic events if M_{23} is considered a kept item.

Fig. 5(c) illustrates the PCH module fault branch. It can be observed that the subevents of intermediate event M_{31} are intermediate event M_{29}, basic event X_{56}, and X_{57}. These two basic events only appear directly in the basic event set of M_{31}, indicating that M_{31} contains two specific basic events.

Based on the aforementioned attribute division method, the corresponding intermediate events of test items are classified

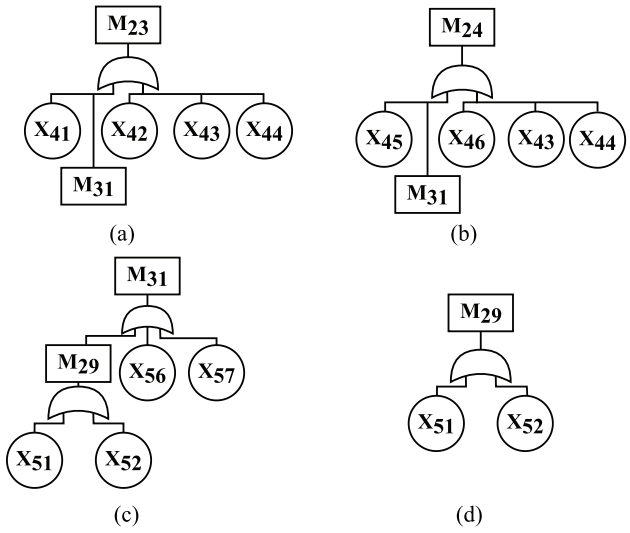

Fig. 5. Four fault branches of the laptop motherboard's fault tree. (a) Microphone fault branch. (b) Speaker fault branch. (c) PCH module fault branch. (d) CPU fault branch.

TABLE IV
FIRST-ROUND ATTRIBUTE DIVISION RESULT BASED ON SPECIFIC BASIC EVENTS

Category	Intermediate event No.
1	23
2	4,10
3	1, 2, 3, 6, 8, 9, 11, 12, 13, 14, 15, 16, 19, 20, 24, 27
4	5, 7, 17, 18, 21, 22, 25, 26, 28

into four categories, containing one, two, three, and four specific basic events individually. Test items of corresponding intermediate events in the first and second categories should be labeled as initially kept items. Additional kept items selected during the second-round attribute selection should be selected from the third category as a priority. X_{12} appears in the basic event sets of multiple intermediate events, while intermediate events in the first and second categories do not contain it. To facilitate the selection of the second round, the corresponding

test item of M_6 is also designated as an initially kept item. The results of the first-round attribute division are shown in Table IV.

2) Imputed Attribute Selection Based on Stratified Sampling: In the second-round selection, attributes are grouped and selected based on the structure of their subevents. Specifically, attributes containing the same shared basic events are grouped together to reflect associations between system modules. To avoid redundant propagation of nonvaluable basic event information and to ensure a balanced distribution of basic event information across attribute sets, stratified sampling is employed for further selection among attribute sets with significant differences.

Conventional attribute selection methods, such as those based on information gain, often rely heavily on the prior probabilities of basic events, which are collected from historical data. As a result, attributes associated with frequently occurring basic events tend to be selected repeatedly, leading to an uneven distribution and high redundancy of valuable basic event information. Moreover, such methods often overlook the structural associations between basic events, making it difficult to capture the intermodule relationships reflected by basic events. In contrast, stratified sampling can divide attributes into sets based on the structural composition of basic events, and samples are drawn from each group proportionally, taking into account both sample size and significance. This approach enhances the balance of valuable basic event information across the selected attributes while retaining correlations between intermediate events, thereby aligning better with the goals of the second-round selection.

In this study, third-category intermediate events from the first round are divided into five sets based on subevents. The first set contains M_3 and M_{27}; these intermediate events include M_{29} and two specific basic events within their subevents. The second set contains M_{11}, M_{12}, M_{13}, M_{14}, M_{15}, and M_{16}; these intermediate events include M_{30} and two specific basic events in their subevents. The third set contains M_8; this intermediate event contains M_7, M_{31}, and two specific basic events in its subevents. The fourth set contains M_{19} and M_{20}; these intermediate events include M_{31}, X_{12}, and two specific basic events in their subevents. The fifth set contains M_1, M_2, M_9, and M_{24}; these intermediate events include M_{31} and two specific basic events in their subevents.

Intermediate events are then randomly selected from each set based on the proportion of intermediate events within that stratum. At last, test items of intermediate events selected in the second-round attribute selection are added to the initially kept items set. Other test items are set as final redundant items, which contain the missing data to be imputed.

After determining the attributes to be imputed, the attribute selection strategy can be established. For a given total number of motherboards (denoted as N) and a total number of test items (denoted as N_t), the testing strategy is represented as $\boldsymbol{S} \in \{0,1\}^{N\times N_t}$, where the strategy for the ith motherboard's jth test item is denoted as $S_{i,j}$. The motherboards with earlier indices are treated as the initial sample set, and all test items for all motherboards in this set will be tested. For subsequent samples following the initial sample set, only kept test items will be tested. Only when the ith motherboard is not in the initial sample set and the jth test item is a redundant test item, will $S_{i,j}$ be set to 0, indicating that the corresponding test item for the corresponding motherboard will not be tested. In all other cases, $S_{i,j}$ will be set to 1, indicating that the corresponding test item for the corresponding motherboard will be tested.

C. Imputed Attribute Mapping Method Based on FTA

In this section, a top–down quantitative analysis method based on the fault tree is proposed for data imputation, which infers the basic event states from intermediate event states. This maps the focus of data imputation from intermediate events to basic events, enhancing the accuracy and information utilization efficiency of the imputation process. In this approach, historical repair data are first statistically analyzed to obtain the prior probabilities of all basic events. Then, using the states of intermediate events, the posterior probabilities of basic events are calculated, enabling the inference of basic event states. The prior probability of a basic event refers to its occurrence probability estimated from historical data. The posterior probability of a basic event is the conditional probability derived by combining its prior probability with intermediate event states.

As illustrated in two cases of intermediate events, the structural and hierarchical information of the fault tree allows for the transfer of basic event information from intermediate events to the basic event state inference, thereby achieving a mapping from intermediate event states to basic event states. In this process, the associations between system modules reflected by shared basic events play a crucial role in the inference, as the inclusion of such associations allows for a more comprehensive capture of the dependencies between basic events.

As shown in Fig. 5(a), assuming that the states of M_{23} and M_{31} are known. If the state of M_{23} is "pass," then the states of X_{41}, X_{42}, X_{43}, and X_{44} are all "pass." Conversely, if the state of M_{23} is "fail," then the states of X_{41}, X_{42}, X_{43}, and X_{44} will fall into the following cases.

1) If the state of M_{31} is "fail," then the states of X_{41}, X_{42}, X_{43}, and X_{44} do not affect M_{23} and cannot be inferred from M_{23}. The posterior probabilities of X_{41}, X_{42}, X_{43}, and X_{44} are determined by the prior probabilities collected from repair data.
2) If the state of M_{31} is "pass," then at least one of the states of X_{41}, X_{42}, X_{43}, and X_{44} is "fail," which falls into the following cases.
    a) One of the states of the four basic events is "fail."
    b) Two of the states of the four basic events are "fail."
    c) Three of the states of the four basic events are "fail."
    d) All of the states of the four basic events are "fail."

Equations (1)–(3) can be obtained from the aforementioned analysis as follows.

Fig. 5(d) shows the CPU fault branch of the motherboard fault tree. Assume the state of M_{29} is known. If the state of M_{29} is "pass," then both the states of X_{51} and X_{52} are "pass."

If the state of M_{29} is "fail," then at least one of the states of X_{51} and X_{52} is "fail," which falls into three cases.

1) The state of X_{51} is "fail," and the state of X_{52} is "pass."
2) The state of X_{51} is "pass," and the state of X_{52} is "fail."
3) The state of X_{51} is "fail," and the state of X_{52} is "fail."

Equations (4) and (5) can be derived from the analysis above in the following manner:

$$P(X_{41}=1,X_{42}=1,X_{43}=1,X_{44}=1|M_{23}=1)=1 \tag{1}$$

$$P(X_j=i|M_{23}=0,M_{31}=0)=P(X_j=i)\quad i=0,1,\ j=41,42,43,44 \tag{2}$$

$$P(X_{41}=i,X_{42}=j,X_{43}=k,X_{44}=l|M_{23}=0,M_{31}=1)=\frac{P(X_{41}=i,X_{42}=j,X_{43}=k,X_{44}=l)}{1-P(X_{41}=1,X_{42}=1,X_{43}=1,X_{44}=1)}\quad i,j,k,l=0,1\wedge i+j+k+l<4 \tag{3}$$

$$P(X_{51}=1,X_{52}=1|M_{29}=1)=1 \tag{4}$$

$$P(X_{51}=i,X_{52}=j|M_{29}=0)=\frac{P(X_{51}=i,X_{52}=j)}{1-P(X_{51}=1,X_{52}=1)}\quad i,j=0,1\wedge i+j<2. \tag{5}$$

After the first-round attribute division, the number of initially kept items was 4. During the top–down quantitative analysis, to facilitate the inference of basic event states, intermediate events M_{29}, M_{30}, and M_{31} should be inferred first. For these intermediate events, at least two attributes containing these intermediate events are required to infer the states accurately.

For M_{31}, all test items in the initially kept items contain it. Concerning M_{30}, one of the test items in the initially kept items contains it, and all test items in the second set of the second-round attribute selection contain it. Regarding M_{29}, none of the test items in the initially kept items contain it, but both test items in the first test item set in the second-round attribute selection do. Consequently, two test items from the first set in the second round will be chosen as the final kept items to simplify the second-round attribute selection process.

The posterior probabilities of basic events are determined through top–down quantitative analysis. Using this method, the states of intermediate events can be translated into the logical operation results of basic event states. Following this imputed attribute mapping process, the focus of data imputation maps from corresponding intermediate events of attributes to basic events, and the imputed data change from missing attributes to unknown basic events. As a result, the proportion of imputed data is reduced. In the attribute selection method, this approach not only emphasizes specific basic events but also retains shared basic events between attributes. Therefore, the mapping from intermediate events to basic events not only reduces the proportion of missing data but also preserves the intermodule associations, improving the accuracy of the imputation.

D. XGBoost-Based Data Imputation

In the process of imputing missing data for unknown basic event states, two major challenges arise: first, the feature space consists solely of discrete variables; second, there exists significant data imbalance in the distribution of basic event states. To address these issues, this article introduces the XGBoost model for predicting unknown basic events. On the one hand, XGBoost, which is built on decision trees, can directly handle discrete feature states, thereby eliminating the need for feature encoding during the preprocessing stage. This capability not only improves modeling efficiency but also enhances prediction accuracy due to its native support for discrete variables. On the other hand, the ensemble framework of XGBoost effectively mitigates the impact of imbalanced data distributions on model performance. Furthermore, its incorporation of regularization techniques improves generalization, while the use of second-order gradient information enhances both computational efficiency and the precision of loss function optimization. Based on these advantages, XGBoost is employed in this article for predicting unknown basic event states.

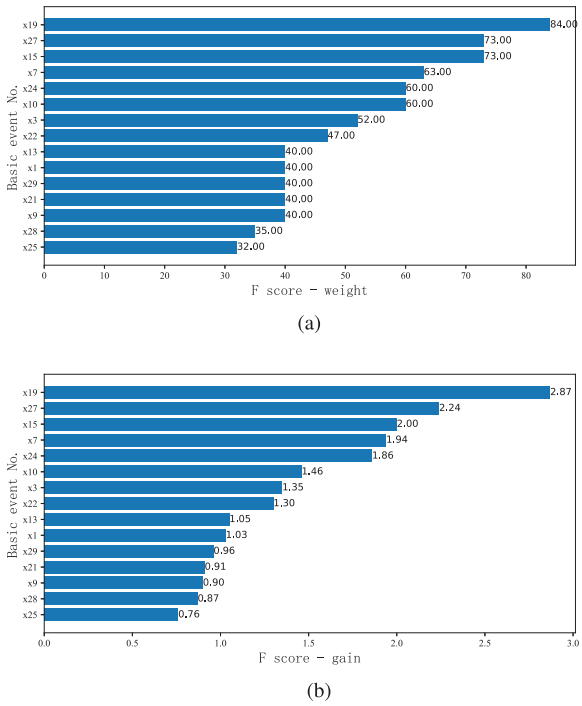

Fig. 6. Feature importance evaluation. (a) Feature importance evaluation—weight. (b) Feature importance evaluation—gain.

Basic events of kept items are designated as an input for XGBoost, while the remaining unknown basic events are designated as an output for XGBoost. The number of base learners in XGBoost in the proposed and comparative methods is 15. The parameter "max_depth" is set to 6. The parameter "learning_rate" is set to 0.02. The parameter "early_stopping_rounds" is set to 5.

Fig. 6 presents the feature importance evaluation through two metrics: weight and gain. Weight quantifies the frequency of a feature being selected as a splitting node across all decision trees, while the gain metric measures the average prediction improvement achieved by splits involving that feature. Both subfigures systematically display the top 15 influential features, each mapped to a basic event. Features x19, x27, x15, x7, and x24 demonstrate the highest importance scores, with x19 emerging as the most critical feature. Conversely, x25 consistently ranks as the lowest-impact feature. The feature

importance analysis establishes a direct correlation between feature selection and system failure mechanisms, thus offering actionable guidance for future method refinement.

E. Evaluation Criteria

To evaluate the total cost during the motherboard functional testing, this article employs the average cost (denoted as $\bar{c}$) as an evaluation criterion, which is defined as the weighted sum of the average testing time (denoted as $\bar{t}$) and the average repair cost (denoted as $\bar{r}$), with the specific calculation method detailed in the following equations:

$$\bar{c} = \bar{t} + \omega \cdot \bar{r} \tag{6}$$

$$\bar{t} = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{N_t} (t_{i,j} | S_{i,j} = 1) \tag{7}$$

$$\bar{r} = \frac{N_B \cdot r}{N} \tag{8}$$

where ω denotes the weight applied in the calculation. The average testing time, denoted as $\bar{t}$, indicates the mean testing duration for each motherboard, calculated from the testing times $t_{i,j}$ of various test items and the corresponding values of the testing strategies. The average repair cost, $\bar{r}$, is derived from the average repair cost of a single defective motherboard (denoted as r), the total number of missing identified defective motherboards N_B, and the overall number of motherboards tested.

IV. Case Study

This section introduces the experiment for testing processes with different yields and analyzes the experimental results. All experiments were performed on a computer running Windows 10, equipped with 16 GB of RAM and a 2.50-GHz CPU. The experimental results of the proposed method are compared with six other methods.

1) *ADO:* This method tests only the selected items determined through the attribute determination method, while other redundant items remain untested and will not be imputed. This method does not involve data imputation.
2) *XGB:* This method identifies test items with low yield as kept items and considers other test items with high yield as redundant items. In contrast, the proposed method uses the attribute determination method. Kept items are identified as the input of XGBoost, while redundant items are identified as the output of XGBoost. This method is hereafter referred to as XGB.
3) *NAM:* Compared to the proposed method, this method suggests eliminating the imputed attribute mapping method based on the fault tree.
4) *RF:* Compared to the proposed method, this method is a modified approach that replaces XGBoost with the random forest (RF).
5) *MICE:* Multivariate imputation by chained equations (MICE) iteratively predicts missing values per variable using other features, generating multiple imputed datasets through cyclical updates. Its random sampling mechanism introduces variation across imputations, with the final results aggregated from these probabilistic imputations. In this article, MICE is implemented to impute unknown basic [38].
6) *SAITS:* The self-attention-based imputation for time-series (SAITS) method learns missing values through weighted combinations of two diagonally masked self-attention (DMSA) blocks. The DMSA architecture explicitly captures both temporal dependencies across time steps and feature correlations, thereby enhancing imputation accuracy while accelerating training convergence. The weighted combination mechanism enables SAITS to dynamically assign weights learned from dual DMSA blocks based on attention maps and missing patterns. In this article, SAITS is implemented to impute unknown basic events [39].

A. Data Generation

To protect commercial confidentiality, this article is not authorized to display real-world manufacturing data directly. Therefore, it utilizes transformed data derived from the original dataset. These transformed data have been authorized for both use and publication. The transformation process strictly adheres to the procedure outlined in the following. The column indices representing intermediate events in the dataset were shuffled, thereby altering their mapping to test items. Similarly, the same shuffling process was applied to the dataset containing basic event states. The transformation process involved solely the shuffling of column indices in the numerical tables of the original dataset, with no additional forms of data transformation. As a result, the overall data distribution remained unchanged, and the transformed data retain the same distribution characteristics as the original dataset. This approach to data transformation does not introduce any bias and has no significant impact on the validity of the method evaluation.

Due to data-sharing restrictions imposed by the collaborating enterprise, the real-world manufacturing data were insufficient to fully validate the proposed method. To overcome this limitation, additional transformed data were synthesized based on the causal relationships among system events identified through FTA in Section III-A and guided by the statistical properties of the original transformed data. The synthesized dataset follows the same distribution to ensure transparency and impartiality and is used solely to augment the original dataset. Collectively, both sets of data are referred to as the transformed data.

Based on the results of the aforementioned FTA, the dataset for the motherboard functional testing comprises 57 basic events, which are associated with 33 intermediate events and further mapped to 28 test items. The detailed correspondence among these events is presented in Section III-A. Basic events are listed in Table III, intermediate events in Table II, and test items in Table I. In the transformed data, the table containing the prior probabilities of basic events consists of 57 columns, each corresponding to one basic event. The specific prior probabilities (denoted as p_d) are provided in Table V.

Hyperparameter tuning for decision tree-based machine learning models is crucial for maximizing model efficacy,

TABLE V
PRIOR PROBABILITIES OF BASIC EVENTS IN FAULT TREE WITH DIFFERENT YIELDS

Symbol	Prior Probabilities with High Yield	Prior Probabilities with Low Yield	Prior Probabilities with Yield Variations		
			No. 1-50,000	No. 50,001-100,000	No. 100,001-150,000
X_1	0.0042	0.0101	0.0071	0.0081	0.0017
X_2	0.0009	0.0027	0.0023	0.0024	0.0029
X_3	0.0027	0.0026	0.0062	0.0103	0.0056
X_4	0.0029	0.0033	0.0016	0.0025	0.0041
X_5	0.0056	0.0013	0.0045	0.0026	0.0044
X_6	0.0020	0.0006	0.0015	0.0020	0.0019
X_7	0.0033	0.0037	0.0079	0.0043	0.0058
X_8	0.0041	0.0046	0.0031	0.0023	0.0043
X_9	0.0084	0.0182	0.0183	0.0121	0.0110
X_{10}	0.0055	0.0053	0.0090	0.0029	0.0085
X_{11}	0.0021	0.0044	0.0076	0.0086	0.0063
X_{12}	0.0007	0.0007	0.0002	0.0022	0.0057
X_{13}	0.0052	0.0043	0.0045	0.0067	0.0019
X_{14}	0.0022	0.0073	0.0091	0.0107	0.0022
X_{15}	0.0067	0.0061	0.0056	0.0063	0.0107
X_{16}	0.0021	0.0028	0.0057	0.0029	0.0043
X_{17}	0.0008	0.0025	0.0009	0.0045	0.0055
X_{18}	0.0060	0.0042	0.0081	0.0042	0.0048
X_{19}	0.0014	0.0063	0.0052	0.0083	0.0035
X_{20}	0.0027	0.0067	0.0084	0.0041	0.0016
X_{21}	0.0071	0.0035	0.0060	0.0100	0.0028
X_{22}	0.0043	0.0024	0.0077	0.0047	0.0083
X_{23}	0.0017	0.0068	0.0054	0.0065	0.0019
X_{24}	0.0028	0.0023	0.0021	0.0129	0.0083
X_{25}	0.0049	0.0077	0.0083	0.0081	0.0014
X_{26}	0.0046	0.0039	0.0091	0.0044	0.0074
X_{27}	0.0019	0.0034	0.0089	0.0065	0.0114
X_{28}	0.0028	0.0072	0.0072	0.0041	0.0115
X_{29}	0.0024	0.0031	0.0060	0.0086	0.0037
X_{30}	0.0069	0.0041	0.0076	0.0062	0.0055
X_{31}	0.0015	0.0076	0.0051	0.0040	0.0062
X_{32}	0.0017	0.0025	0.0037	0.0045	0.0041
X_{33}	0.0014	0.0022	0.0095	0.0111	0.0028
X_{34}	0.0023	0.0038	0.0031	0.0068	0.0019
X_{35}	0.0038	0.0061	0.0104	0.0047	0.0020
X_{36}	0.0015	0.0068	0.0051	0.0060	0.0107
X_{37}	0.0047	0.0034	0.0091	0.0063	0.0082
X_{38}	0.0042	0.0039	0.0053	0.0021	0.0041
X_{39}	0.0076	0.0045	0.0022	0.0088	0.0113
X_{40}	0.0043	0.0083	0.0076	0.0065	0.0042
X_{41}	0.0040	0.0021	0.0105	0.0129	0.0075
X_{42}	0.0061	0.0036	0.0025	0.0020	0.0038
X_{43}	0.0035	0.0004	0.0049	0.0019	0.0039
X_{44}	0.0078	0.0069	0.0171	0.0103	0.0147
X_{45}	0.0036	0.0046	0.0063	0.0059	0.0101
X_{46}	0.0006	0.0027	0.0070	0.0057	0.0044
X_{47}	0.0039	0.0066	0.0036	0.0052	0.0018
X_{48}	0.0085	0.0042	0.0052	0.0018	0.0039
X_{49}	0.0014	0.0029	0.0025	0.0017	0.0018
X_{50}	0.0020	0.0041	0.0041	0.0075	0.0041
X_{51}	0.0003	0.0007	0.0036	0.0028	0.0024
X_{52}	0.0025	0.0028	0.0017	0.0006	0.0022
X_{53}	0.0003	0.0036	0.0029	0.0022	0.0019
X_{54}	0.0001	0.0005	0.0041	0.0015	0.0004
X_{55}	0.0002	0.0049	0.0006	0.0011	0.0018
X_{56}	0.0010	0.0002	0.0004	0.0004	0.0001
X_{57}	0.0007	0.0063	0.0005	0.0014	0.0006

primarily categorized into three groups: general parameters, booster parameters, and task parameters. For XGBoost implementations, general parameters include base learner configurations, while task parameters govern learning objectives, training data specifications, and evaluation metrics. Booster parameters demonstrate the most significant impact on performance, encompassing learning rate(learning_rate), number of base learners (n_estimators), maximum tree depth (max_depth), minimum child node weight (min_child_weight), loss reduction threshold (gamma), class balancing coefficient (scale_pos_weight), subsampling ratio (subsample), and regularization weights (alpha for L1 and lambda for L2 regularization).

Following conventional practice, this study employs grid search for systematic hyperparameter optimization, implemented through the integrated GridSearchCV module in scikit-learn 8.0. To address the severe data imbalance, oversampling and data concatenation were applied during preprocessing, with subsequent model evaluation adopting the ROC-AUC metric for comprehensive performance assessment.

The grid search protocol initiates with defining parameter search spaces, prioritizing accelerated convergence through an initial learning rate setting of learning_rate =0.2. Subsequent optimization follows a sequential hierarchy: first determining optimal n_estimators within [5, 10, 15, 20, 25, 30] through cross-validated fitting and then refining max_depth (range: 2–10) and min_child_weight (range: 1–6), followed by gamma adjustment and subsample tuning.

After finalizing each parameter combination, the fit function of GridSearchCV is called to iteratively determine the optimal value for each hyperparameter. Evaluation metrics include mean_test_score, std_test_score, and rank_test_score, representing average scores during multiple testing, variance between test scores, and the scores' ranks. Definitive parameter combinations are extractable via the best_params function in GridSearchCV, which is n_estimators =15, min_child_weight = 3, and max_depth =6 in the proposed method. This structured approach ensures systematic exploration of parameter interactions while maintaining computational efficiency.

When considering the characteristics of the missing identified defective motherboards, all results of kept items and corresponding basic events show "pass." Therefore, for these motherboards, all basic event states in the input are "pass." To enhance prediction accuracy, the proposed method introduces random values that align with the prior probabilities of basic events. The number of columns added to the input corresponds to the number of unknown basic events. The values in these columns are randomly set to "pass" or "fail" based on the respective prior probabilities. The proposed method oversamples defective boards in the initial 50000 boards, which are utilized to train the XGBoost model. This method can increase the number of defective samples in the training set.

B. Results Analysis of the Test Outcome With High Yield

In this section, the superiority of the proposed method is validated through a high-yield testing process, with the basic event prior probabilities presented in the second column of Table V.

It should be noted that the number of test items in the first, third, and fourth sets of the second-round attribute selection is relatively low. Assuming all test items in these sets are selected proportionally, three test items should be selected from the second set, and two should be selected from the fifth set proportionally. In this situation, the number of kept items is 14; thus, the appropriate number of kept items is set as 14 for now. To compare the impact of the number of kept items, 14 is set as the median and will be compared with 11, 16, and 18. Use the first 50000 mainboards as the initial sample set and test all test items for each sample. Subsequent samples will only test the kept items. The results of comparing different numbers of kept items under a testing process with high yield are shown in Table VI, where "dppm" represents the defects per million. The problem-solving time of all methods is presented in Table VII.

As evidenced in Table VI, reducing the number of kept items increases the defect level. Balancing average cost and defect level requires careful equilibrium, where average test time, the primary factor affecting average cost, correlates directly with the number of kept items. Key observations reveal: when decreasing kept items from 14 to 11, defect level surges by 94 dppm; conversely, increasing to 16 items yields merely a 19-dppm improvement. Further escalation to 18 items achieves an 80-dppm reduction. Cost-efficiency suggests optimal kept item selection should either minimize items while meeting defect thresholds or prioritize configurations with significant attenuation in the decrease of defect level. The marginal 80-dppm improvement from 16 to 18 items proves less impactful than the 94-dppm penalty from 14 to 11 items, while variation between 14 and 16 items remains negligible. Consequently, 14 kept items emerge as the optimal configuration. In the subsequent experiments, the number of kept test items for each method is set to 14.

When the number of kept items is set to 14, defect levels decrease by 51.02% compared to attribute determination only (ADO) and 29.91% compared to XGB. These findings highlight that test strategies relying on a single model are subject to the constraints of that model. Compared to no attribute mapping (NAM), the proposed method shows a 26.00% decrease in defect levels, demonstrating the effectiveness of the imputed attribute mapping method based on FTA. When comparing the proposed method to RF, the defect level reduction is 26.00%. This finding indicates that the XGBoost model is more appropriate than the random forest for data imputation in the motherboard functional test. When comparing the proposed method to MICE and SAITS, the defect level reductions are 25.31% and 23.15%. This result proves the usability of the proposed method under highly unbalanced situations like the motherboard functional test.

When it comes to average cost, there is a 5.64% decrease compared to ADO, a 2.39% decrease compared to XGB, a 1.98% decrease compared to NAM, a 1.98% decrease compared to RF, a 1.91% decrease compared to MICE, and a 1.70% decrease compared to SAITS. This decrease seems small because the average test time affects the average cost the most. However, the numbers of kept items determined by different methods are always different in actual situations. The average cost of the proposed method with different numbers of kept items shows a significant reduction.

Regarding problem-solving time, the proposed method is inferior to NAM and XGB but superior to RF, MICE, and SAITS. The ADO method does not utilize a machine learning model and does not require a calculation after the determination of the imputed attribute. Thus, it does not have a problem-solving time. MICE and SAITS cannot meet the real-time adjustment problem due to the long problem-solving time.

C. Results Analysis of the Test Outcome With Low Yield

In this section, the proposed method is validated through a testing process characterized by a relatively low yield. The corresponding basic event prior probabilities are listed in the third column of Table V, and the results are presented in Table VIII.

TABLE VI
EVALUATION CRITERIA WITH HIGH YIELD OF DIFFERENT METHODS

Number of Kept Items	Method	Average Test Time/s	Defect Level		**Average Cost**	
			Value/s	Reduction/%	Value/dppm	Reduction/%
11	ADO	110.99	647	48.53	1368.10	10.11
	XGB		540	38.33	1315.91	6.54
	NAM		447	25.50	1282.06	4.08
	RF		447	25.50	1282.06	4.08
	MICE		447	25.50	1282.06	4.08
	SAITS		480	30.62	1282.73	4.13
	Proposed method		333	\	1229.81	\
14	ADO	141.26	488	**51.02**	1588.32	**5.64**
	XGB		341	29.91	1535.40	2.39
	NAM		323	26.00	1528.92	1.98
	RF		323	26.00	1528.92	1.98
	MICE		320	25.31	1527.84	1.91
	SAITS		311	23.15	1524.60	1.70
	Proposed method		**239**	\	**1498.68**	\
16	ADO	161.45	367	40.05	1746.57	3.02
	XGB		301	26.91	1722.81	1.69
	NAM		260	15.30	1708.05	0.84
	RF		260	15.39	1708.05	0.84
	MICE		257	14.40	1706.97	0.78
	SAITS		245	10.20	1702.65	0.53
	Proposed method		220	\	1693.65	\
18	ADO	182.40	293	42.19	1948.93	3.82
	XGB		260	38.95	1907.89	1.75
	NAM		200	20.00	1900.69	1.38
	RF		200	20.00	1900.69	1.38
	MICE		200	20.00	1899.61	1.33
	SAITS		201	20.39	1896.37	1.16
	Proposed method		160	\	1874.41	\

TABLE VII
PROBLEM-SOLVING TIME OF DIFFERENT METHODS

Method	ADO	XGB	NAM	RF	MICE	SAITS	Proposed method
Time/s	\	**1.00**	1.83	4.74	171.45	557.31	3.90

TABLE VIII
EVALUATION CRITERIA WITH LOW YIELD OF DIFFERENT METHODS

Method	Defect Level		**Average Cost**	
	Value/s	Reduction/%	Value/dppm	Reduction/%
ADO	701	**43.94**	1665.00	**6.66**
XGB	560	29.82	1614.24	3.72
NAM	467	15.85	1580.76	1.69
RF	467	15.85	1580.76	1.69
MICE	463	15.12	1579.32	1.60
SAITS	453	13.25	1575.72	1.37
Proposed Method	**393**	\	**1554.12**	\

The defect level of the proposed method has increased from 239 to 393 dppm. The defect level of the proposed method is 43.94% lower than ADO, 29.82% compared to XGB, 15.85% compared to NAM, 15.85% compared to RF, 15.12% compared to MICE, and 13.25% compared to SAITS. When it comes to average cost, there is a 6.66% decrease compared to ADO, a 3.72% decrease compared to XGB, a 1.69% decrease compared to NAM, a 1.69% decrease compared to RF, a 1.60% decrease compared to MICE, and a 1.37% decrease compared to SAITS.

TABLE IX
EVALUATION CRITERIA WITH LONG-TERM YIELD VARIATION OF DIFFERENT METHODS

Method	Defect Level		**Average Cost**	
	Value/s	Reduction/%	Value/dppm	Reduction/%
ADO	697	**46.92**	1663.56	**7.07**
XGB	634	41.64	1640.88	5.79
NAM	626	40.89	1638.00	5.63
RF	626	40.89	1638.00	5.63
MICE	624	40.71	1637.28	5.58
SAITS	620	40.32	1635.84	5.50
Proposed Method -Static	512	27.73	1596.96	3.20
Proposed Method -Adaptation	**370**	\	**1545.84**	\

It can be seen that the defect level of the proposed method increased compared to the defect level with high yield, but it still achieved outstanding performance compared to other methods. The proposed method's reduction in the defect level compared to other methods is consistent with that under high yield.

D. Results Analysis of the Test Outcome With Long-Term Yield Variation

To simulate long-term operational changes, the experimental dataset comprises 500000 motherboards, divided into three units of 150000 boards each. Within each unit, the basic event prior probabilities vary sequentially according to Columns 4 through 6 of Table V, with the final 50000 boards reverting

to the values in Column 4. This configuration results in nine shifts in prior probabilities, effectively extending beyond the variations captured in the original transformed data.

Comparative methods persistently tested all items on the initial 50000 motherboards but subsequently only kept items without model retraining. An additional comparative method "proposed method-static" is introduced, maintaining the initial imputation model trained on the first 50000 motherboards without subsequent updates. In contrast, the adaptive implementation of the proposed method conducts full retesting and model retraining every 150000 motherboards, generating three specialized imputation models, respectively, applicable to motherboards 50001–150000, 200001–300000, and 350001–500000.

In real-world manufacturing, the motherboard functional test procedure comprises 12–18 test stations handling an average testing time of 141.26 s per motherboard, as documented in Table VI, meaning that the minimum test interval is 7.85 s. In real-time adjustment analysis, Table VII highlights that the proposed method achieves a problem-solving time of 3.90 s, outperforming the minimum test interval (7.85 s). In contrast, the problem-solving times for MICE (171.45 s) and SAITS (557.31 s) both exceed the minimum test interval. Therefore, the proposed method meets the real-time adjustment time requirements in industrial production, while the problem-solving times of the comparison methods MICE and SAITS do not meet the real-time constraints. As evidenced in Table IX, the adaptive approach significantly reduces defect level compared to static methods, with periodic retraining effectively counteracting performance degradation caused by long-term yield variation.

The experiments mentioned above demonstrate that the proposed method, based on FTA and XGBoost, is effective. The proposed method is superior to all other comparative methods in reducing the cost of the motherboard functional tests and defect levels. This is particularly true when dealing with a production process with long-term yield variations or a relatively low yield. The reduction in defect levels is evident and consistent. The experimental results demonstrate that the proposed method is superior to ADO, XGB, NAM, RF, MICE, and SAITS in reducing defect levels and average cost. The experimental results also show that the proposed method is more efficient than RF, MICE, and SAITS in problem-solving. When the number of kept items is set to an optimal value of 14 in this article, the proposed method significantly reduces average costs while maintaining an acceptable defect level.

V. Conclusion

This article addresses the limitations of the existing data imputation methods, which inadequately consider the influence of correlation information between system modules on the input information of data imputation models, resulting in suboptimal imputation performance and limited cost reduction in functional testing. By integrating system fault mode analysis with data-driven modeling, a low-cost data imputation method based on FTA and XGBoost is proposed. First, an improved fault tree oriented toward the motherboard functional testing process, which considers faults in nonbus data transmissions, is established. Next, attributes requiring data imputation are determined based on the fault tree structure, considering the correlations between attributes and basic event information. Furthermore, a hybrid data- and mechanism-driven top–down quantitative analysis method for the fault tree is introduced. This method infers basic event states from intermediate event states, mapping the focus of data imputation from intermediate events to basic events. Finally, the tree-based XGBoost model is utilized to impute unknown data. Experiments conducted on transformed real-world manufacturing data demonstrate that effective testing strategies can be dynamically derived under varying conditions using the proposed method. Testing costs are reduced by up to 5.64% in high-yield scenarios, 6.66% in low-yield scenarios, and 7.07% during long-term yield fluctuations. Furthermore, the defect level is reduced by up to 51.02%. Given that laptop motherboards and other types of motherboards typically feature a modular design and clear structural organization, the proposed FTA method is able to extract causal fault relationships across various motherboard types, demonstrating its broader applicability beyond just laptops.

Declaration of Competing Interest

The authors declare that they have no relevant financial or non-financial competing interests that could have appeared to influence the work reported in this article.

References

[1] I. Pomeranz, "Functional design-for-testability for functional test sequences," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 43, no. 12, pp. 4852–4859, Dec. 2024.

[2] M. Liu, F. Ye, X. Li, K. Chakrabarty, and X. Gu, "Board-level functional fault identification using streaming data," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 40, no. 9, pp. 1920–1933, Sep. 2021.

[3] R. Pan, Z. Zhang, X. Li, K. Chakrabarty, and X. Gu, "Black-box test-cost reduction based on Bayesian network models," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 40, no. 2, pp. 386–399, Feb. 2021.

[4] T. Song, Z. Huang, and A. Yan, "Machine learning classification algorithm for VLSI test cost reduction," *Integration*, vol. 87, pp. 40–48, Nov. 2022.

[5] L. Huang and T. Song, "VLSI test through an improved LDA classification algorithm for test cost reduction," *Microelectron. J.*, vol. 125, Jul. 2022, Art. no. 105461.

[6] A. Pavlidis, M.-M. Louërat, E. Faehn, A. Kumar, and Haralampos-G. Stratigopoulos, "SymBIST: Symmetry-based analog and mixed-signal built-in self-test for functional safety," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 68, no. 6, pp. 2580–2593, Jun. 2021.

[7] C. Yang, "Parallel–series multiobjective genetic algorithm for optimal tests selection with multiple constraints," *IEEE Trans. Instrum. Meas.*, vol. 67, no. 8, pp. 1859–1876, Aug. 2018.

[8] I. Polian et al., "Exploring the mysteries of system-level test," in *Proc. 29th Asian Test Symp. (ATS)*, Nov. 2020, pp. 1–6.

[9] E. A. Dri, G. M. Peretti, and E. A. Romero, "A low-cost test strategy based on transient response method for embedded reconfigurable filters," *Int. J. Electron.*, vol. 108, no. 4, pp. 664–683, Apr. 2021.

[10] F. Deng, M. Jin, C. Liu, M. Liserre, and W. Chen, "Switch open-circuit fault localization strategy for MMCs using sliding-time window based features extraction algorithm," *IEEE Trans. Ind. Electron.*, vol. 68, no. 10, pp. 10193–10206, Oct. 2021.

[11] P. Bai, Y. Kang, K. Wang, Y. Zhao, and S. Dong, "Modelling and optimizing motherboard functional testing in laptop manufacturing," *J. Syst. Sci. Complex.*, vol. 37, no. 6, pp. 2406–2423, Sep. 2024.

[12] M. Liu, R. Pan, F. Ye, X. Li, K. Chakrabarty, and X. Gu, "Fine-grained adaptive testing based on quality prediction," *ACM Trans. Design Autom. Electron. Syst.*, vol. 25, no. 5, pp. 1–25, Sep. 2020.

[13] X. Miao, Y. Wu, L. Chen, Y. Gao, and J. Yin, "An experimental survey of missing data imputation algorithms," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 7, pp. 6630–6650, Jul. 2022.

[14] N. Karmitsa, S. Taheri, A. Bagirov, and P. Mäkinen, "Missing value imputation via clusterwise linear regression," *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 4, pp. 1889–1901, Apr. 2022.

[15] Y. Yuan, Y. Zhang, B. Wang, Y. Peng, Y. Hu, and B. Yin, "STGAN: Spatio-temporal generative adversarial network for traffic data imputation," *IEEE Trans. Big Data*, vol. 9, no. 1, pp. 200–211, Feb. 2023.

[16] H. Kim, D.-E. Lim, and S. Lee, "Deep learning-based dynamic scheduling for semiconductor manufacturing with high uncertainty of automated material handling system capability," *IEEE Trans. Semicond. Manuf.*, vol. 33, no. 1, pp. 13–22, Feb. 2020.

[17] L. Sun and F. You, "Machine learning and data-driven techniques for the control of smart power generation systems: An uncertainty handling perspective," *Engineering*, vol. 7, no. 9, pp. 1239–1247, Sep. 2021.

[18] D. Adhikari et al., "A comprehensive survey on imputation of missing data in Internet of Things," *ACM Comput. Surveys*, vol. 55, no. 7, pp. 1–38, Jul. 2023.

[19] S. Nikfalazar, C.-H. Yeh, S. Bedingfield, and H. A. Khorshidi, "Missing data imputation using decision trees and fuzzy clustering with iterative learning," *Knowl. Inf. Syst.*, vol. 62, no. 6, pp. 2419–2437, Jun. 2020.

[20] H. Huang, H. Wang, and M. Sun, "Incomplete data classification with view-based decision tree," *Appl. Soft Comput.*, vol. 94, Sep. 2020, Art. no. 106437.

[21] C.-F. Tsai and Y.-H. Hu, "Empirical comparison of supervised learning techniques for missing value imputation," *Knowl. Inf. Syst.*, vol. 64, no. 4, pp. 1047–1075, Apr. 2022.

[22] H. Jeong, H. Wang, and F. P. Calmon, "Fairness without imputation: A decision tree approach for fair prediction with missing values," in *Proc. AAAI Conf. Artif. Intell.*, Jun. 2022, vol. 36, no. 9, pp. 9558–9566.

[23] J. S. Rhodes, A. Cutler, and K. R. Moon, "Geometry- and accuracy-preserving random forest proximities," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 45, no. 9, pp. 10947–10959, Sep. 2023.

[24] Q.-T. Phan, Y.-K. Wu, and Q.-D. Phan, "Enhancing one-day-ahead probabilistic solar power forecast with a hybrid transformer-LUBE model and missing data imputation," *IEEE Trans. Ind. Appl.*, vol. 60, no. 1, pp. 1396–1408, Jan. 2024.

[25] A. I. Khalyasmaa, M. D. Senyuk, and S. A. Eroshenko, "Analysis of the state of high-voltage current transformers based on gradient boosting on decision trees," *IEEE Trans. Power Del.*, vol. 36, no. 4, pp. 2154–2163, Aug. 2021.

[26] X. Chen, D. Yu, X. Fan, L. Wang, and J. Chen, "Multiclass classification for self-admitted technical debt based on XGBoost," *IEEE Trans. Rel.*, vol. 71, no. 3, pp. 1309–1324, Sep. 2022.

[27] S. Yun, X. Yang, B. Wang, C. Ji, B. Lin, and X. Bai, "Research on the optimization method of SMT process parameters based on improved PSO algorithm," *IEEE Trans. Compon., Packag., Manuf. Technol.*, vol. 14, no. 6, pp. 1113–1122, Jun. 2024.

[28] H. Li, K. Lyu, Z. Yong, W. Xiaolong, J. Qiu, and G. Liu, "Multichannel parallel testing of intermittent faults and reliability assessment for electronic equipment," *IEEE Trans. Compon., Packag., Manuf. Technol.*, vol. 10, no. 10, pp. 1636–1646, Oct. 2020.

[29] L. von Rueden et al., "Informed machine learning—A taxonomy and survey of integrating prior knowledge into learning systems," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 1, pp. 614–633, Jan. 2023.

[30] W. Luo, O. Wei, and H. Wan, "SATMCS: An efficient SAT-based algorithm and its improvements for computing minimal cut sets," *IEEE Trans. Rel.*, vol. 70, no. 2, pp. 575–589, Jun. 2021.

[31] S. Zhou, L. Ye, S. Xiong, and J. Xiang, "Reliability analysis of dynamic fault trees with priority-AND gates based on irrelevance coverage model," *Rel. Eng. Syst. Saf.*, vol. 224, Aug. 2022, Art. no. 108553.

[32] C. Zhou, Q. Chang, H. Zhao, M. Ji, and Z. Shi, "Fault tree analysis with interval uncertainty: A case study of the aircraft flap mechanism," *IEEE Trans. Rel.*, vol. 70, no. 3, pp. 944–956, Sep. 2021.

[33] M. Tripathi, L. K. Singh, S. Singh, and P. Singh, "A comparative study on reliability analysis methods for safety critical systems using Petri-nets and dynamic flowgraph methodology: A case study of nuclear power plant," *IEEE Trans. Rel.*, vol. 71, no. 2, pp. 564–578, Jun. 2022.

[34] J. F. W. Peeters, R. J. I. Basten, and T. Tinga, "Improving failure analysis efficiency by combining FTA and FMEA in a recursive manner," *Rel. Eng. Syst. Saf.*, vol. 172, pp. 36–44, Apr. 2018.

[35] B. Sun, X. Fan, L. Li, H. Ye, W. van Driel, and G. Zhang, "A reliability prediction for integrated LED lamp with electrolytic capacitor-free driver," *IEEE Trans. Compon., Packag., Manuf. Technol.*, vol. 7, no. 7, pp. 1081–1088, Jul. 2017.

[36] J. Leng, Z. Chen, W. Sha, Z. Lin, J. Lin, and Q. Liu, "Digital twins-based flexible operating of open architecture production line for individualized manufacturing," *Adv. Eng. Informat.*, vol. 53, Aug. 2022, Art. no. 101676.

[37] Y. Li, K. Wang, Y. Kang, Y. Zhao, and P. Bai, "Board-level functional test selection based on fault tree analysis," in *Proc. 6th Int. Symp. Auto. Syst. (ISAS)*, Jun. 2023, pp. 1–6.

[38] S. van Buuren and K. Groothuis-Oudshoorn, "MICE: Multivariate imputation by chained equations in R," *J. Stat. Softw.*, vol. 45, no. 3, pp. 1–67, Dec. 2011.

[39] W. Du, D. Côté, and Y. Liu, "SAITS: Self-attention-based imputation for time series," *Expert Syst. Appl.*, vol. 219, Jun. 2023, Art. no. 119619.