RESEARCH-ARTICLE

Signal Feature Coverage and Testing for CPS Dataflow Models

EZIO BARTOCCI, Vienna University of Technology, Vienna, Vienna, Austria

LEONARDO MARIANI, University of Milano-Bicocca, Milan, MI, Italy

DEJAN NIČKOVIĆ, Austrian Institute of Technology, Vienna, Austria

DRISHTI YADAV, Vienna University of Technology, Vienna, Vienna, Austria

Published: 18 August 2025
Online AM: 22 January 2025
Accepted: 06 January 2025
Revised: 27 November 2024
Received: 14 May 2024

Signal Feature Coverage and Testing for CPS Dataflow Models

EZIO BARTOCCI, TU Wien, Vienna, Austria
LEONARDO MARIANI, University of Milano-Bicocca, Milan, Italy
DEJAN NICKOVIC, AIT Austrian Institute of Technology, Vienna, Austria
DRISHTI YADAV, TU Wien, Vienna, Austria

Design of cyber-physical systems (CPS) typically involves dataflow modeling. The structure of dataflow models differs from the traditional software, making standard coverage metrics not appropriate for measuring the thoroughness of testing. To address this limitation, this article proposes *signal feature coverage* as a new coverage metric for systematically testing CPS dataflow models. We derive signal feature coverage by leveraging signal features. We developed a testing framework in Simulink, a popular dataflow modeling and simulation environment, that automates the generation and execution of test cases based on the defined coverage metric. We evaluated the effectiveness of our approach by carrying out experiments on five Simulink models tested against ten Signal Temporal Logic specifications. We compared our coverage-based testing approach to adaptive random testing, falsification testing, output diversity-based approaches, and testing using MathWorks' Simulink Design Verifier. The results demonstrate that our coverage-based testing approach outperforms the conventional techniques regarding fault detection capability.

CCS Concepts: • **Software and its engineering → Software testing and debugging**;

Additional Key Words and Phrases: Cyber-Physical Systems, Simulink models, Coverage criteria, Testing, Signal Temporal Logic (STL)

ACM Reference format:
Ezio Bartocci, Leonardo Mariani, Dejan Nickovic, and Drishti Yadav. 2025. Signal Feature Coverage and Testing for CPS Dataflow Models. *ACM Trans. Softw. Eng. Methodol.* 34, 7, Article 199 (August 2025), 37 pages.
https://doi.org/10.1145/3714467

1 Introduction

Testing is essential in ensuring the reliability and safety of **Cyber-Physical Systems (CPS)**. The embedded software industry is progressively leaning towards **Model-Based Development (MBD)** methodologies to streamline their development processes and deliver high-quality software

This work has been supported by the Doctoral College Resilient Embedded Systems, which is run jointly by the TU Wien's Faculty of Informatics and the UAS Technikum Wien. This work has been also supported by the Centro Nazionale HPC, Big Data e Quantum Computing (PNRR CN1 spoke 9 Digital Society and Smart Cities); and the Engineered MachinE Learning-intensive IoT systems (EMELIOT) national research project, which has been funded by the MUR under the PRIN 2020 program (Contract 2020W3A5FY).
Authors' Contact Information: Ezio Bartocci, TU Wien, Vienna, Austria; e-mail: ezio.bartocci@tuwien.ac.at; Leonardo Mariani, University of Milano-Bicocca, Milan, Italy; e-mail: leonardo.mariani@unimib.it; Dejan Nickovic, AIT Austrian Institute of Technology, Vienna, Austria; e-mail: Dejan.Nickovic@ait.ac.at; Drishti Yadav (corresponding author), TU Wien, Vienna, Austria; e-mail: drishti.yadav@tuwien.ac.at.

ACM 1557-7392/2025/8-ART199
https://doi.org/10.1145/3714467

components more efficiently. MBD involves creating models as the primary artifacts during the development process. These models serve as abstract representations of the software system and its components, capturing various aspects such as behavior, structure, and interactions.

In practice, engineers commonly employ commercial toolchains and dataflow modeling languages, such as MathWorks Simulink [42], to prototype safety-critical CPS during concept design phase. These tools enable the engineers to model and analyze CPS and to automatically compile them into executable code for embedded systems [47, 59, 60]. This approach facilitates efficient development and validation processes for CPS, contributing to the reliability and safety of the final product.

Timely fault detection is crucial in the CPS development cycle, as rectifying faults post-hardware integration is prohibitively costly. As safety-critical CPS becomes increasingly complex [57], thorough testing of CPS dataflow models becomes increasingly pronounced, demonstrating the need of robust test generation strategies capable of promptly identifying and addressing potential problems. In the realm of embedded software testing, practitioners commonly evaluate the efficacy of test suites by gauging their code coverage and their ability to reveal faults.

Structural coverage metrics (e.g., statement and branch coverage) are extensively used in research [23, 69] and industry [28]. Structural measures can capture how much a test suite exercises the elements of a system, highlighting potential inadequacies in the test cases. However, numerous studies indicate that relying solely on structural coverage criteria may not be sufficient for detecting faults in software models and programs [24, 27, 50, 63]. This is particularly true for standard metrics applied to CPS dataflow models. In fact, due to the interconnected nature of the elements in a dataflow model, executing a test case to cover one element might easily trigger the execution of many other elements. For instance, in our experiments, 7.6 random test cases were on average enough to cover more than 90% of the signals in our tested CPS models, and in one case, a single test was sufficient to cover the signals in the tested model.[1] As a result, measuring structural coverage provides little insight about the effectiveness of a test suite, potentially leading to overlooked defects in the system. To effectively test CPS models, it is essential to exploit more refined and advanced notions of coverage that are finely tuned to capture the nuanced behaviors and interactions within these models.

In this article, we address this challenge by proposing a new concept of coverage, that is *signal feature coverage*, which requires exercising the *internal signals* of a CPS dataflow model in terms of their various time and frequency *domain features*. Contrary to classical coverage metrics ranging in the $[0-100\%]$ interval, signal feature coverage is purposely not designed to saturate, ranging in the $[0, \infty]$ domain. It captures the fact that it is always possible to exercise software more thoroughly. Similar to classical structural metrics, increasing coverage can be expensive and one must find a good trade-off between costs and benefits.

Based on this notion of coverage, we also designed **Feature-Coverage Testing (FCT)**, a search-based test generation strategy tailored to generate test suites that maximize the signal feature coverage. Besides, we instantiate our FCT approach with different sets of target signals and their target features. Our first variant, the **Comprehensive Feature-Coverage Testing (FCT-C)** strategy, is dedicated to thoroughly testing the system by maximizing the coverage of all features across all signals within a dataflow model. Furthermore, we investigate a testing scenario involving CPS dataflow models evaluated against pre-defined formal specifications expressed in **Signal Temporal Logic (STL)** [35]. In this context, our second variant of FCT, referred to as

[1] A signal is considered *covered* if, during execution, it flows or propagates along its associated line or connection in the model. In this sense, it can be viewed as a type of structural coverage, as it pertains to the activation of certain structural elements of the system, i.e., lines/connections between components.

Specification-directed Feature-Coverage Testing (FCT-S), aims to maximize the coverage of only those features and signals that have a significant impact on the outputs of the system and the STL specifications that it is expected to satisfy.

Contributions. In a nutshell, our article provides the following contributions:

(1) We define *signal feature coverage*, a new coverage criterion designed to better assess the thoroughness of test suites for CPS dataflow models.
(2) We present *FCT*, a novel test case generation strategy that can be instantiated in two variants to thoroughly exercise CPS dataflow models.
(3) We experimented signal feature coverage and FCT with five Simulink models and ten STL properties, demonstrating that the generation of test cases according to signal feature coverage can increase mutation score approximately by 26% (on an average) compared to the best-performing method among all the tested state-of-the-art methods (including **Adaptive Random Testing (ART)**, **Falsification Testing (FT)**, **Output Diversity (OD)**-based testing, and testing using MathWorks' **Simulink Design Verifier (SDV)**).

Our data package and all the details of our evaluation results are available at https://gitlab.com/DrishtiYadav/fct_master.

The remainder of the article is organized as follows. Section 2 presents the relevant background. Section 3 presents the signal feature coverage criterion, and the coverage-based test generation algorithm. Section 4 presents our evaluation on five industrial CPS dataflow Simulink models. Section 5 presents the related work. Section 6 provides final remarks.

2 Preliminaries

This section provides necessary background information on signals [52], whose values are processed in CPS models, STL [35], a logic-based formalism suitable to specify safety requirements of CPS models, and CPS dataflow models.

2.1 Signals and Systems

Definition 2.1 (Signal [52]). A signal W is a function $W : \mathbb{T} \rightarrow \mathbb{D}$, where $\mathbb{T}$ represents the time domain s.t. $\mathbb{T} \subseteq \mathbb{R}_{\geq 0}$. For a *time-bounded* signal, $\mathbb{T}$ is in the form of $[0, T]$ where the time horizon $T \in \mathbb{R}_{>0}$ is a positive real. If $\mathbb{D} = \mathbb{B}$, the signal W is a Boolean signal, taking either true or false values. On the other hand, if $\mathbb{D} = \mathbb{R}$, the signal is real-valued.

Definition 2.2 (Trace [52]). A *trace*, denoted by $\mathbf{W} : \mathbb{T} \rightarrow \mathbb{D}_1 \times \ldots \times \mathbb{D}_n$, is a collection of n signals s.t. $\forall t \in \mathbb{T}, \mathbf{W}(t) = (W_1(t), W_2(t), \ldots, W_n(t))$. We can view $\mathbf{W}$ as an execution trace representing the evolution of a continuous-time system with n variables.

Definition 2.3 (System [52]). A system Π, with an m-dimensional input and an n-dimensional output, is a function that maps a given input trace $\mathbf{W}_{In} : \mathbb{T} \rightarrow \mathbb{D}^m$ to an output trace $\mathbf{W}_{Out} : \mathbb{T} \rightarrow \mathbb{D}^n$.

2.2 STL

STL [35] is a well-established temporal logic to specify linear-time properties of mixed-analog signals. The use of STL is prevalent in the analysis of programs within CPS that involve interactions with physical entities exhibiting continuous dynamics.

Definition 2.4 (STL Syntax [35]). The STL syntax is defined as

$$\Phi := true \mid p \mid f(x) > 0 \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid \Phi_1 \mathcal{U}_I \Phi_2,$$

where p and x are Boolean and real-valued variables,[2] f is a real-valued function from signal domain to $\mathbb{R}$, and $I \subseteq \mathbb{R}_{\geq 0}$ is an arbitrary time interval of non-negative real numbers. $\mathcal{U}$ is the Until operator implying that Φ_2 becomes true at some time within the interval I and Φ_1 must remain *true* until Φ_2 becomes *true*. We can derive other Boolean and temporal operators as follows: $\Phi_1 \vee \Phi_2 \equiv \neg(\neg\Phi_1 \wedge \neg\Phi_2)$, $\Diamond_I\, \Phi \equiv true\ \mathcal{U}_I\, \Phi$, $\Box_I\, \Phi \equiv \neg\Diamond_I\, \neg\Phi$, where $\Diamond$ is the *eventually* operator and $\Box$ is the *always* operator. Untimed notations, such as $\Box$, $\Diamond$, and $\mathcal{U}$, are used as shorthand for $\Box_{[0,\infty)}$, $\Diamond_{[0,\infty)}$ and $\mathcal{U}_{[0,\infty)}$, respectively. Intervals in STL formulas are often expressed with arithmetic expressions; e.g., $\Diamond_{=a}$ for $\Diamond_{[a]}$ and $\Diamond_{\leq a}$ for $\Diamond_{[0,a]}$.

STL formulas are evaluated according to their robustness (quantitative) finitary semantics [18].

Definition 2.5 (STL Robustness Semantics [18, 22]*).* The robustness of an STL formula Φ w.r.t. a trace $\mathbf{W}$ at time $t \in \mathbb{T}$, denoted by $\rho(\Phi, \mathbf{W}, t)$, is defined inductively as

$$
\begin{aligned}
\rho(true, \mathbf{W}, t) &= +\infty \\
\rho(f(x) > 0, \mathbf{W}, t) &= f(\mathbf{W_x(t)}) \\
\rho(p, \mathbf{W}, t) &= \begin{cases} +\infty, & \text{if } W_p(t) \\ -\infty, & \text{otherwise} \end{cases} \\
\rho(\neg\Phi, \mathbf{W}, t) &= -\rho(\Phi, \mathbf{W}, t) \\
\rho(\Phi_1 \wedge \Phi_2, \mathbf{W}, t) &= \min\left(\rho(\Phi_1, \mathbf{W}, t), \rho(\Phi_2, \mathbf{W}, t)\right) \\
\rho(\Phi_1\, \mathcal{U}_I\, \Phi_2, \mathbf{W}, t) &= \sup_{t' \in ((t \oplus I) \cap \mathbb{T})} \Big(\min\left(\rho(\Phi_2, \mathbf{W}, t'), \inf_{t'' \in (t, t')} (\rho(\Phi_1, \mathbf{W}, t''))\right)\Big).
\end{aligned}
$$

The robustness metric ρ maps a trace $\mathbf{W}$ and a formula Φ to a real number r. We say that the formula Φ is satisfied (violated) by $\mathbf{W}$ when r is positive (negative). The value of r indicates the degree of satisfaction or violation of Φ by $\mathbf{W}$. We observe that the STL semantics assumes continuous signals with values defined at every time point in $\mathbb{T}$. However, in practice, simulation engines generate finite representations of signals consisting of finite sequences of (time, value) pairs. We use in this article piecewise-constant (also known as sample-and-hold) interpolation to provide continuous-time interpretation to the signals generated by simulation.

2.3 CPS Dataflow Models

A CPS dataflow model $\mathcal{M}$ formally represents the CPS behavior. As described in [61], it is defined as a quadruple $\langle \mathcal{V}, \mathcal{B}, \mathcal{C}, \mathcal{S} \rangle$ with:

- —a set of variables $\mathcal{V} = \{\mathcal{V}_I, \mathcal{V}_H, \mathcal{V}_O\}$ where $\mathcal{V}_I$, $\mathcal{V}_H$ and $\mathcal{V}_O$ denote the input, hidden/internal, and output variables respectively,
- —a set of blocks $\mathcal{B}$ which process the data acquired via their *input ports*, transmit the output data via their *output ports*, using lines also known as *connections*,
- —a set of connections $\mathcal{C}$ between the blocks $\mathcal{B}$, and
- —a set of signals $\mathcal{S}$, s.t. each $s \in \mathcal{S}$ associates a connection $c \in \mathcal{C}$ with a variable $v \in \mathcal{V}$.

CPS dataflow languages usually enable the creation of *hierarchical* models using *Subsystem* and *Model Reference* features. Figure 1 shows a hierarchical CPS dataflow model with $\mathcal{M}_1^1$ as the *top-level* model. For a model $\mathcal{M}_i^k$, the subscript represents the model number while the superscript indicates the hierarchy level. $\mathcal{M}_1^1$ has two child models $\mathcal{M}_2^2$ and $\mathcal{M}_3^2$, both at hierarchy level 2.

[2]We associate p and x to a multi-dimensional signal $W : \mathbb{T} \to \mathbb{D}^n$, where $\mathbb{T} = [0, d)$ is a finite time domain of duration d, and we denote by W_p and W_x the projection of W to its p and x components, respectively.

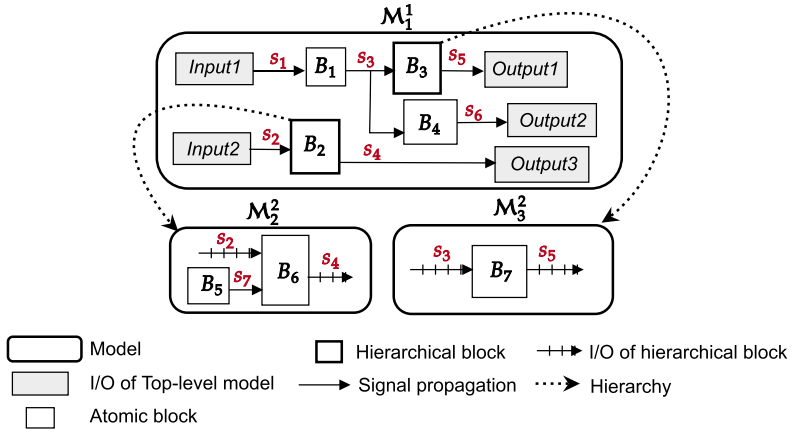


Fig. 1. A hierarchical CPS dataflow model with two inputs and three outputs. s_1 and s_2 are input signals. s_4, s_5, and s_6 are output signals, while the remaining are internal signals.

A dataflow model may include feedback-loops (exhibiting circular relationships between signals), and the conditional execution of subsystems, for instance based on actions (such as *if-else* statements, *switch* cases), enabled states, and triggers.

Concrete CPS dataflow modeling environments (e.g., Simulink) typically provide simulation capabilities (with user-defined sampling).[3] We denote an input vector U for the model $\mathcal{M}$ s.t. U includes all the values for each variable in $\mathcal{V}_I$.

We define $\mathfrak{S}$ as the simulation function that takes U and $\mathcal{M}$ as inputs, simulates $\mathcal{M}$ against U and returns a trace O as the model simulation output. $O(U, \mathcal{M})$ represents the sequence of system states that evolve with discrete timesteps from time 0 to time T, where the simulation time $T \in \mathbb{R}_{>0}$. The trace O is composed of system (input/internal/output) signals, where each signal is a sequence of (timestamp, value) pairs.

3 Signal Feature Coverage

We now introduce the notion of a *signal feature* and identify a set of such useful features (Section 3.1). We then propose *signal feature coverage* (Section 3.2) as a test adequacy metric for CPS dataflow models. We finally present an algorithm to generate test cases that maximize the introduced signal feature coverage (Section 3.3). We also instantiate the feature-coverage-based test generation technique with the coverage criterion that focuses on signals and features that are the most likely to have a significant impact on the system (Section 3.3.2).

3.1 Signal Features

Signal features are statistical, spectral, and various other measurable properties. Other measurable signal properties cover a broad spectrum of attributes that can be quantified to describe a signal comprehensively. Beyond statistical and spectral features, these properties may include temporal characteristics (such as rise time, decay time, settling time) [12], phase information (including phase shifts and phase angle), frequency-related aspects (like bandwidth), energy metrics (such as total signal energy or power), and waveform attributes (such as pulse width and waveform symmetry).

[3]Sampling indicates the size of the timesteps, which can be either *fixed-step* or *variable-step*. Unless otherwise specified, we employ a fixed-length sampling for all the systems considered in this article.

Table 1. The Basic Statistics Feature Set [58]

Feature	Mathematical Formula
Mean	$\mu = \frac{1}{k}\sum_{i=1}^{k} u_i$
SD	$\sigma = \sqrt{\frac{1}{k-1}\sum_{i=1}^{k} \lvert u_i - \mu\rvert^2}$
RMS	$\mathbf{u}_{RMS} = \sqrt{\frac{1}{k}\sum_{i=1}^{k} \lvert u_i\rvert^2}$
SF	$\mathbf{u}_{SF} = \frac{\mathbf{u}_{RMS}}{\frac{1}{k}\sum_{i=1}^{k} \lvert u_i\rvert}$

In this section, we identify a standard set of signal features employed in the CPS application domain and signal processing. When identifying and collecting relevant signal features, we focused on ensuring their meaningfulness and utility for comprehensive testing and analysis by applying the following criteria: (i) how changes in the features affect the system's response and stability, (ii) variations in these features should help in detecting deviations from expected behavior or identifying faults, (iii) features should cover various aspects of the system's dynamics, including both normal and extreme operating conditions, and (iv) features should be consistent with those used in previous studies and across various application domains.

To collect features, we conducted a thorough review of existing literature to identify those that have been effectively utilized in similar contexts. We identified a total of ten signal features that can provide valuable information about the behavior of a system. The selection of these signal features is supported by their established significance in signal processing, system analysis, and fault detection [9, 31, 33, 56, 58, 68]. They have been widely used in various domains, including communications, biomedical applications, and control systems. By incorporating these features in our testing approach for Simulink models of CPS, we aim to capture and evaluate critical aspects of signal behavior that can contribute to adequate system testing, fault diagnosis, and performance evaluation. We group the selected signal features into four feature sets: basic statistics, higher-order statistics, impulsive metrics, and frequency features. In the following, we use the bold letter $\mathbf{u}$ for a finite-length signal composed of k samples s.t. $\mathbf{u} = (u_1, u_2, ..., u_k)$.

3.1.1 Basic Statistics (Four Features). The basic statistics are used to measure the dispersion of a signal [58]. Table 1 lists the features included in this feature set. They consist of the mean, SD, **Root Mean Square (RMS)**, and **Shape Factor (SF)**.

3.1.2 Higher-Order Statistics (Two Features). The higher-order statistics measure the symmetry and flatness of a signal [58], and provide insights about the system behavior through the third moment (skewness) and fourth moment (kurtosis) of the signal [41]. Table 2 summarize these metrics.

3.1.3 Impulsive Metrics (Three Features). The impulsive metrics are used to evaluate peak changes in the signal, including peak value, impulse factor, and crest factor [41]. Table 2[4] provides a brief description of each metric along with its mathematical representation.

3.1.4 Frequency Features (One Feature). The concept of frequency is well-defined for sinusoidal signals, but real-world signals are often more complex. Non-stationary signals pose challenges for frequency analysis, leading to the emergence of the concept of *instantaneous frequency* [10, 11]. Instantaneous frequency captures the time-varying nature of signals and plays a key role in diverse applications such as seismic analysis, radar systems, communications, and biomedical research.

[4]Note that in Table 1 and Table 2, the $|\cdot|$ operator represents the absolute value mathematical operation.

Table 2. The Higher-Order Statistics and Impulsive Metrics of a Signal

	Feature	Description	Mathematical Formula
Higher-order statistics	Kurtosis	It quantifies the distribution shape (length of the tails) of a signal [56].	$\mathbf{u}_{kurt} = \frac{\frac{1}{k}\sum_{i=1}^{k}(u_i-\mu)^4}{\left[\frac{1}{k}\sum_{i=1}^{k}(u_i-\mu)^2\right]^2}$
	Skewness	It measures the asymmetrical spread of a signal about its mean value [41, 56].	$\mathbf{u}_{skew} = \frac{\frac{1}{k}\sum_{i=1}^{k}(u_i-\mu)^3}{\left[\frac{1}{k}\sum_{i=1}^{k}(u_i-\mu)^2\right]^{\frac{3}{2}}}$
Impulsive metrics	Peak value	It is the peak or maximum absolute value of the signal [58].	$\mathbf{u}_p = \max_{1\le i\le k} \lvert u_i \rvert$
	Impulse factor	"It presents the impulsive behavior and the multitude of peaks" [32] in the signal.	$\mathbf{u}_{IF} = \frac{\mathbf{u}_p}{\frac{1}{k}\sum_{i=1}^{k}\lvert u_i \rvert}$
	Crest factor	It indicates how extreme the peaks are in the signal [41].	$\mathbf{u}_{crest} = \frac{\mathbf{u}_p}{\mathbf{u}_{RMS}} = \frac{\mathbf{u}_p}{\sqrt{\frac{1}{k}\sum_{i=1}^{k}\lvert u_i \rvert^2}}$

In our study, we include the feature "*Peak Instantaneous frequency*" in the frequency feature set to analyze and understand the associated signals more effectively.

Mathematically, the instantaneous frequency $f_{inst}(t)$ is estimated as the derivative of the phase of the analytic signal of the input signal [10, 11], i.e.,

$$f_{inst}(t) = \frac{1}{2\pi}\frac{d\phi}{dt},$$

where ϕ is the phase of the analytic signal $\mathbf{u}_A$ of the input signal $\mathbf{u}$. In general, the *phase* of a signal refers to the angle component of its complex representation, which varies over time. While the phase itself is not an instantaneous measure, the derivative of the phase with respect to time gives us the instantaneous frequency, which reflects the rate of change of the phase at each moment in time. The analytic signal $\mathbf{u}_A$ of $\mathbf{u}$ is computed by taking the *hilbert transform* of $\mathbf{u}$. The *Peak Instantaneous frequency*, denoted by $\mathbf{u}_{PIF}$ is then computed as the maximum value of $f_{inst}(t)$.

3.2 Signal Feature Coverage

Let us consider an internal signal s of a CPS dataflow model $\mathcal{M}$ and a feature $\mathbf{x}$ of the signal s, denoted as $\mathbf{x}^s$. We assume that a feature $\mathbf{x}^s$ has to be exercised within a range R bounded by its minimum and maximum values denoted by min_R and max_R, respectively. Let $X^s = \{x_1^s, x_2^s, \ldots, x_n^s\}$ be a set of n observations of the feature $\mathbf{x}^s$.

To measure how thoroughly X^s is exercising the feature $\mathbf{x}$ of s, we measure how well the values in X^s *uniformly* and *densely* sample the values of the feature. This approach gives a reasonable confidence over the capability of the test suite to consider a good range of behaviors for the chosen feature and signal.

To introduce this measure, we employ a simple automatic binning algorithm to partition the range R into N bins $R_1, \ldots R_N$ of equal size. For instance, if $R = [0, 10]$ and $N = 4$, the resulting bins are $[0, 2.5)$, $[2.5, 5)$, $[5, 7.5)$, $[7.5, 10]$.[5] We refer to these N bins as the *Standard bins*.

Given a set of samples X^s and a range R split in N bins $R_1, \ldots R_N$, we define the notion of bin occupancy and bin coverage as follows.

[5]We conventionally consider the last bin as closed on both ends.

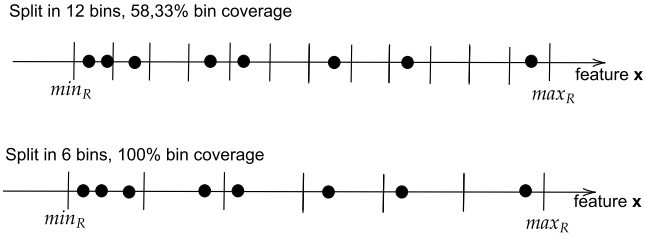


Fig. 2. Sample bin coverage of a feature.

Definition 3.1 (Bin Occupancy). A bin R_i is occupied, if $\exists x_i^s \in X^s$ such that $x_i^s \in R_i$. We say that

$$Occ(R_i) = \begin{cases} 1, & \text{if } R_i \text{ is occupied} \\ 0, & \text{otherwise} \end{cases}$$

More simply, bin occupancy refers to whether a specific bin in the range is occupied by at least one sample from the set of test samples. It acts as a binary indicator for each bin, where a value of 1 signifies that the bin is occupied and 0 signifies that it is not.

Definition 3.2 (Bin Coverage). The *Bin Coverage*, denoted by β, is then defined as

$$\beta = \frac{\#\,Standard\,Bins\,Covered}{\#\,Standard\,Bins} = \frac{\sum_{i=1}^{N} Occ(R_i)}{N}$$

Unlike bin occupancy, which focuses on individual bins, bin coverage aggregates the occupancy of all bins, providing a metric that quantifies how thoroughly the range of signal features has been explored. By aiming for higher bin coverage, we ensure that the test cases span a wide variety of signal feature values, contributing to a more comprehensive evaluation of the system under test. Thus, while bin occupancy addresses individual bins, bin coverage captures the overall extent of coverage across all bins.

Figure 2 shows two sample bin coverage cases for an identical number of samples. The bottom case shows the case of eight samples achieving 100% bin coverage with a split in six bins. The top case shows the case of the same eight samples only achieving $\frac{7}{12} = 58.33\%$ bin coverage for a division in 12 bins.

To measure how well a feature is densely and uniformly sampled, we compute the *finest division* in bins of its domain that is covered by the available samples. Intuitively, the more a feature is well exercised throughout its domain, the more it should be possible to consider a finer-grained division that is well covered by the available samples. As a measure of signal feature coverage, we use the number of bins in the finest division. For instance, if a set of ten samples of a feature can cover up to five standard bins (i.e., bins of the same length), the achieved coverage is five.

Achieving higher signal feature coverage levels is increasingly harder with additional samples. For instance, covering four bins requires a minimum of four well-distributed values, while covering a division in five or six bins, requires better distribution of samples.

This measure of coverage satisfies some unusual but useful properties. In particular, the coverage is measured as a natural number, which (a) discretizes the domain capturing the incremental coverage levels that can be achieved, and (b) is unbounded following the intuition that a tester can always add additional evidence that better exercises a system. Indeed, an empirical **Research Question (RQ)** is finding valuable coverage levels that can represent a useful trade-off between cost and results (which is not a unique question of this coverage metric, but it is also common to bounded coverage metrics that are seldom addressed until reaching 100% coverage).

More rigorously, given a set of samples X^s, we define the signal feature coverage of a feature $\mathbf{x}$ of a signal s as follows.

Definition 3.3 (Individual Signal Feature Coverage).

$$\zeta_{\mathbf{x}^s} = \arg\max_{N} (\beta \geq 0.95) \tag{1}$$

Note that here, to be practical and accommodate for intervals that might be particularly hard or impossible to cover, we require the samples to cover a significant portion of the standard bins, defined as 95% of the bins in the partition.

We see the signal feature as mainly a multi-optimization problem, that is, test suites should target the individual features and signals, and coverage reports should detail the results per signal. However, it is also possible to derive an aggregated coverage measure for an entire system, for instance considered the average or minimum feature coverage value obtained for a system. In our analysis, we calculate the coverage measure for a system by determining the *average* of *individual signal feature coverage* values across the target signals and target features within the system.

Definition 3.4 (Signal Feature Coverage). In typical dataflow system models, there is often an extensive array of signals, each potentially containing numerous features that testers may wish to investigate beyond those initially outlined in Section 3.1. Let S be the set of all signals of the dataflow model of a system and F be the set of all features. Given $S' \subseteq S$ as the subset of signals that a tester wishes to investigate, and for each signal $s \in S'$, there is a corresponding subset $F'_s \subseteq F$ of features that are applicable to that signal, the *signal feature coverage* is computed as follows:

$$\zeta_{S'F'_s} = \frac{1}{|S'|} \sum_{s \in S'} \frac{1}{|F'_s|} \sum_{\mathbf{x} \in F'_s} \zeta_{\mathbf{x}^s} \tag{2}$$

To tailor the coverage assessment according to specific testing requirements, we provide two instantiations of the meta-definition of $\zeta_{S'F'_s}$ with different subsets of signals and features. For a comprehensive analysis and to attain a thorough understanding of the system, a tester may wish to investigate the coverage of all features across all signals within a system model. This variant of signal feature coverage is termed as the "*Comprehensive Signal Feature Coverage*" (ζ_{comp}), computed with $S' = S$ (all signals) and $F'_s = F$ (all features).

Alternatively, testers may opt to narrow the focus and seek to compute signal feature coverage for a specific subset of signals and their corresponding subset of features, which can be advantageous for system testing. These subsets are typically determined by the testers and may require domain knowledge or thorough testing. A relevant case consists of selecting only the features that may significantly impact on a property ϕ under test. This variant of our coverage criterion is referred to as "ϕ*-driven Signal Feature Coverage*" (ζ_ϕ), focusing on the examination of *important* features $F_{s,\phi}$ of *important* signals S_ϕ that substantially influence the satisfaction of ϕ. ζ_ϕ is $\zeta_{S'F'_s}$ with $S' = S_\phi$ and $F'_s = F_{s,\phi}$.

3.3 Testing for Signal Feature Coverage

We now introduce *FCT*, a test generation strategy that aims to maximize the *signal feature coverage* for S' signals and their features F'_s. We formulate the test generation process as a single-objective optimization problem of maximizing the *signal feature coverage* of the target features of the target signals of a system.

Signal Feature Coverage-Based Test Generation Problem

Input: a dataflow model $\mathcal{M}$ with S' signals and features F'_s of each signal.
Problem: Find $\mathcal{TS}$ s.t. $\zeta_{S'F'_s}$ is maximum.

This maximization problem has the goal of achieving a test suite $\mathcal{TS}$ with the best combination of test cases $\mathbf{t} \in \mathcal{TS}$ that effectively covers the target features of the target signals of the system. We instantiate the test generation procedure using Simulink as our CPS dataflow modeling and simulation environment, and **Particle Swarm Optimization (PSO)** [30], a population-based meta-heuristic optimization algorithm, for maximizing the signal feature coverage by iteratively improving a test suite with regard to the coverage metric. PSO is suitable for operating in irregular search spaces. Its strong optimization ability, coupled with its ease of implementation, has made PSO a popular choice for solving real-world problems, including but not limited to system-level testing [62] and specification-based testing [4].

We employ the standard PSO algorithm as introduced in the original work [30], utilizing the default PSO parameters. In our PSO-based test generation technique, a test $\mathbf{t}$ represents a particle in the swarm, and the swarm corresponds to a test suite. In our instantiation, we defined fitness ($\zeta_{S'F'_s}$) at the swarm level rather than at the individual particle level. This decision aligns with our objective of maximizing $\zeta_{S'F'_s}$, derived from the entire test suite rather than a single test case.

Like classical PSO, we start with an initial population which consists of the particle positions and particle velocities: particle *position* indicates the test case, particle *velocity* indicates the velocity associated to a test case. Note that we represent a test case as either (a) a numeric vector $[pos_0, \ldots, pos_N]$, with each value pos_i representing a characteristic of the system's input signal. For instance, in the **Aircraft Elevator Control System (AECS)** model, the system input comprises the *pilot command*, characterized by its amplitude Amp and frequency $Freq$. Thus, the test case for the AECS model is denoted as $t_{AECS} = [Amp, Freq]$, where both Amp and $Freq$ assume numeric values within their valid ranges, or (b) a vector of control points $[pos_0, \ldots, pos_N]$, with each value pos_i representing a control point for parameterized inputs.

We then compute the fitness $\zeta_{S'F'_s}$ as defined in Section 3.2. Upon performing the coverage computation, we obtain a set of bins partitioning the feature values of the signal s associated with the test cases in the population. To maximize feature coverage, we aim to have at least one element in each bin while promoting diversity in the feature values. To achieve this, we update the test cases using the PSO update equations. Equation (3) models the update of the particles' *velocity*, while Equation (4) the update of particles' *position*.

$$V_i^{t+1} \leftarrow wV_i^t + c_1 r_1 (P_{best_i} - pos_i^t) + c_2 r_2 (G_{best_i} - pos_i^t) \tag{3}$$

$$pos_i^{t+1} \leftarrow pos_i^t + V_i^{t+1} \tag{4}$$

In PSO, a group of particles, each representing a potential solution, move through the solution space toward an optimal/near-optimal solution. Each particle i in the swarm has a position pos_i and a velocity V_i. In particular, particle movements are influenced by two main factors: their own best-known position (personal best, P_{best}) and the best-known position of the entire swarm (global best, G_{best}). These positions in the search-space guide the particles to explore and converge toward promising regions, optimizing the overall search process. Some parameters govern the *velocity* update (from the current iteration t to the next iteration $t+1$): the *inertial weight* w regulates the effect of the previous particle velocity on the current one, a *cognitive coefficient* c_1 representing the influence of the particle's personal best, a *social coefficient* c_2 representing the influence of the global best, and two random numbers r_1, r_2 in [0,1] controlling the impact of the personal and

global best on the particle's velocity update. The random numbers r_1, r_2 introduce randomness into the update process and add a degree of exploration to the search. The *position* update depends on the previous position and the current velocity.

The PSO update equations guide the tests to explore and converge toward promising regions, optimizing the overall search process. In cases with multiple feature values in a bin, we select a random test case from that bin, treat it as the global best, and update the remaining test cases in that bin using the standard PSO update equations. After updating the test cases, the algorithm recomputes the swarm's fitness. The updated test suite is accepted as an optimal solution if the value of $\zeta_{S'F'_s}$ improves. The process is repeated until the budget is exhausted.

Various testing requirements can influence the generation of tests for FCT, as discussed below.

3.3.1 Comprehensive Feature-Coverage Testing, FCT-C. When the focus of testing spans the entire set of signals S and features F, the strategy employed is known as FCT-C. The objective of FCT-C is to maximize the coverage of all signal features, termed as comprehensive signal feature coverage ζ_{comp}, that is, FCT-C is obtained by instantiating the PSO problem described above on ζ_{comp}.

3.3.2 Specification-directed Feature-Coverage Testing, FCT-S. In practical settings, testing activities often aim to ensure the system's compliance with functional, non-functional, and safety specifications. Particularly in cases where the system's adherence to safety specifications is paramount, testing is guided by formal specifications expressed in STL. In such scenarios, we refer to the FCT approach as FCT-S. Within FCT-S, the coverage scope is confined to a meaningful subset of features within the internal signals that influence the specified safety requirements. In our instantiation of FCT-S, we restrict signal feature coverage to the *features of the internal signals that significantly impact* the robustness (and hence also the satisfaction/violation) of the STL specifications.[6]

There is no unique way of measuring the impact of a feature on the robustness. In our work, we propose to estimate impact as the correlation between the values of a feature and the robustness of the simulation trace w.r.t. the STL property, and to empirically discover correlations through a bivariate statistics-based procedure, which was revealed to be effective according to our empirical results.

Algorithm 1 describes our significant feature identification procedure, which iteratively reduces (i) the signal population by considering its correlation with the output, and (ii) the number of features by considering its correlation with the robustness w.r.t. the STL property. It works in three stages. First (lines 6–9), the procedure generates and executes tests that can provide evidence of correlation among signals and features. Second (lines 10–15), it then iteratively adds randomly generated test cases to the existing test suite and checks correlation to incrementally reduce the set of the significant signals and significant features. The gradual decrease in the number of signals is achieved by discarding 20% (based on the discard threshold) of the least correlated signals in each iteration. Third (line 16), it removes the redundant internal signals (i.e., the significant signal variables whose values are implied by other significant signal variables) and their associated features to obtain the final reduced set of significant signals and features, i.e., $\hat{\mathcal{V}}_H$ and $\hat{\mathbf{X}}$, respectively.

More in details, the algorithm starts with a model $\mathcal{M}$, a property ϕ, and a set of signal features `Feat` to be investigated by the procedure. In our experiments, we consider the ten features presented in Section 3.1. The algorithm starts by randomly generating an initial test suite (line 6). Then, it uses

[6]We note that the impact of a feature can be also directly evaluated on the model outputs in a specification-less setting. This has however limited interest for CPS dataflow models that are commonly tested against known requirements.

Algorithm 1: Feature Identification Procedure: Identifying Significant Features of Significant Signal Variables that Impact ϕ

Input : $\mathcal{M}$ - A CPS dataflow model.
ϕ - An STL specification of $\mathcal{M}$.
Feat - A set of signal features.
Output: $\hat{\mathcal{V}}_H$ - Significant signal variables.
$\hat{\mathbf{X}}$ - Significant signal features.

1 $SimData$ = [ ]; $Robustness$ = [ ]
2 $\hat{\mathbf{X}}$ = [ ]; **CORR** = [ ]; **corr** = [ ]
3 $\theta_1 = 0.99$; // initialize redundancy threshold
4 $\theta_2 = 0.5$; // initialize correlation threshold
5 $\theta_3 = 0.2$; // initialize discard threshold
6 $\mathcal{TS}$ = RandomTestSuite($\mathcal{M}$)
7 $SimData, Robustness \leftarrow$ Monitor($\mathcal{TS}, \mathcal{M}, \phi$)
8 $\hat{\mathcal{V}}_H \leftarrow$ FindTopS($SimData, \theta_3$) ; // identify internal signals correlated with output
9 $\hat{\mathbf{X}} \leftarrow$ FindF($\hat{\mathcal{V}}_H, SimData$, Feat, $Robustness, \theta_2$) ; // identify features that correlate with robustness
10 **for** *q times* **do**
11 $\mathcal{TS} = \mathcal{TS} \cup \mathcal{J}$; // add tests to the test suite
12 $SimData, Robustness \leftarrow$ Monitor($\mathcal{TS}, \mathcal{M}, \phi$)
13 $\hat{\mathcal{V}}_H \leftarrow$ FindTopS($SimData, \theta_3$)
14 $\hat{\mathbf{X}} \leftarrow$ FindF($\hat{\mathcal{V}}_H, SimData$, Feat, $Robustness, \theta_2$)
15 **end for**
16 $\hat{\mathcal{V}}_H, \hat{\mathbf{X}} \leftarrow$ Remove($SimData, \theta_1, \hat{\mathcal{V}}_H, \hat{\mathbf{X}}$) ; // discard redundant internal signals and associated features
17 **return** $\hat{\mathcal{V}}_H, \hat{\mathbf{X}}$

the function Monitor that simulates the model against each test in the test suite and computes the robustness ρ (according to Definition 2.5) of each resulting simulation trace w.r.t. STL property ϕ.

To identify the significant signals, the algorithm uses FindTopS subroutine (Algorithm 2) that computes the correlation coefficients between each internal signal in $\mathcal{V}_H$ and the output signals to measure their strength of association (lines 1–7). It then rejects a small fraction of the internal signals that exhibit lower strength of correlation with the system outputs based on the discard threshold (line 8). The resulting set of internal signals (i.e., $\hat{\mathcal{V}}_H$) are identified as the signals that show strong association with the system output.

In the next step, the procedure uses the subroutine FindF (Algorithm 3) that identifies the features of each internal signal $v \in \hat{\mathcal{V}}_H$ that have significant impact on the robustness of ϕ. For each $v \in \hat{\mathcal{V}}_H$, the algorithm computes the feature values for all the features in the feature set Feat (line 2), and finds the correlation coefficient between the signal feature values and the robustness ρ (line 3). It marks a feature as relevant if the feature exhibits a strong impact on ρ, i.e., the absolute value of the correlation coefficient exceeds the correlation threshold θ_2. A measure greater than 50% is chosen to identify a subset of significant features $\hat{\mathbf{X}}$ that have either strong or very strong impact on the robustness values.

Algorithm 2: The Subroutine FindTopS()

```
for each v ∈ V_H do
    for each item t ∈ TS do
        corr_v = Correlation(v, V_{O_t})
        CORR = CORR ∪ corr_v
    end for
    corr = corr ∪ CORR
end for
V̂_H ← Reject(corr, V_H, θ_3) ;          // discard internal signals exhibiting low
 correlation with output
return V̂_H
```

Algorithm 3: The Subroutine FindF()

```
for each v ∈ V̂_H do
    A = Feature(Simdata_v, Feat)
    [γ] ← Select(A, Robustness, θ_2) ;    // identify features that correlate with
     robustness
    X̂ = X̂ ∪ [γ]
end for
return X̂
```

Subsequently, the procedure incrementally incorporates a series of test cases into the current test suite (line 11) and recalculates the robustness (line 12) and correlation values to further reduce the cardinality of the significant internal signals set and the significant features set (lines 13 and 14).

As the final step, the procedure further reduces the number of significant signals (and features) by discarding the redundant internal signals and the features associated with the redundant signals (line 16). To identify redundant internal signals, the procedure uses the function Remove that computes the correlation coefficients between the current set of significant internal signals in the model according to the well-known Pearson method and removes the ones that carry redundant information. More specifically, whenever the correlation coefficient between any two internal variables v_1 and v_2 exceeds the redundancy threshold θ_1, we remove one of the two variables from further analysis and the features associated with the redundant signal variables. The resulting set of significant internal signals/features are identified as the signals/features targeted by FCT-S.

In this work, we used three key thresholds: redundancy, correlation, and discard. These thresholds were carefully selected to balance the need for reducing the signal population while retaining the most relevant information. The discard threshold was set at 0.2 to ensure that only a small fraction of internal signals, which exhibit lower correlation with system outputs, are rejected in each iteration. This gradual reduction helps to iteratively refine the signal set without losing significant information. A higher discard threshold could aggressively reduce the signal population but risks omitting critical signals.

For the correlation threshold, we chose a value of 0.5 to focus on features that have a strong or very strong impact on robustness values, ensuring that only the most influential features are retained. However, varying this threshold could lead to identifying a different set of significant features and could impact the test results. To identify and remove redundant signals, we set the redundancy

threshold at 0.99, ensuring that only signals with *very high* correlation are considered redundant. Lowering this threshold would increase the risk of discarding signals that, while correlated, still contain unique and valuable information. Varying any of these thresholds can yield different results in terms of the selected signals and features, and the overall performance of the algorithm.

Test case generation is finally defined by instantiating the PSO problem described above on $\zeta_{\hat{\mathcal{V}}_H \hat{\mathbf{X}}}$.

4 Empirical Evaluation

We empirically evaluate the proposed coverage criteria and its associated coverage-based test generation strategies in comparison to multiple state-of-the-art test generation strategies. In particular, we investigate the following RQs:

RQ1-Coverage. To what extent do state-of-the-art test generation approaches exercise features of CPS Simulink models according to signal feature coverage? The first RQ aims at studying whether signal feature coverage is a non-trivial problem. This RQ investigates *how well existing test generation solutions perform according to signal feature coverage*. If state-of-the-art solutions already generate test suites with high feature coverage, there is little point in studying how to generate test suites that maximize coverage. To answer this question, we compare the effectiveness of three test generation approaches against our feature coverage test generation solution to discover how thoroughly signals' features can be covered and identify any gap in existing test generators. Section 4.1.1 describes the considered test generation solutions. Furthermore, we investigate how variations in budget size influence the coverage achieved. This examination allows us to understand the efficiency of resource allocation in improving the coverage of signal features.

RQ2-Fault Revealing Capability. Are test suites achieving high feature coverage also revealing more faults in CPS Simulink models? This RQ exploits mutation testing to *compare the effectiveness of feature coverage-driven test generation techniques*, namely FCT-C and FCT-S, to state-of-the-art testing techniques. We describe the mutation testing process in Section 4.1.4 and we consider the test generation solutions described in Section 4.1.1.

4.1 Experimental Setup

We now describe the compared test generation solutions, our execution platform, experimental subjects, and mutations.

4.1.1 Test Generation. Our study considers two sets of test generation strategies: the two variants of signal feature coverage (FCT-C and FCT-S) and the eight state-of-the-art test generation solutions for CPS Simulink models described below (ART, OD-VB, OD-FB, SDV-EC, SDV-MCDC, FT-Ψ-TaLiRo, FT-S-TaLiRo, FT-Confidence).

- —FCT-C is the proposed FCT generation algorithm instantiated to cover all targets (all signals and all features) to determine feature-coverage levels that can be feasibly reached.
- —FCT-S is the strategy that only targets significant features of the important signals ($\hat{\mathcal{V}}_H$ and $\hat{\mathbf{X}}$) that are obtained using the Feature Identification Algorithm described in Section 3.3.2.
- —*ART*[7] is a black-box test generation approach focused on generating diverse test cases "by maximizing distances between the test inputs" [15, 34]. It is usually considered as a baseline due to its simplicity and effectiveness.
- —*OD* is a black-box test generation approach that seeks to produce test cases with highly diverse outputs [34, 45, 46]. OD exploits a search-based strategy that starts with an initial

[7]Given that ART typically demonstrates superior performance over random testing (RT) in terms of both failure-detection capability and program-based coverage thoroughness [14, 36], we have not considered RT in the comparison.

test suite and iteratively improves it by making small changes to the test cases to maximize the OD. The technique examines neighboring test cases at each iteration and selects the one that significantly increases OD. The process continues until no further improvement can be made. In this article, we compare with two notions of OD: **Vector-Based (VB)** and **Feature-Based (FB)**, as proposed in [46]. While the *VB* concept is directly applied to output signal vectors, the *FB* concept operates over feature vectors of output signals. This involves describing the *output* signals using their *value* and their *first and second derivative*, which characterize distinguishable signal shapes.

—MathWorks' Simulink Design Verifier (SDV) is utilized to generate tests aimed at achieving model coverage. In this article, we compare with two types of model coverage with SDV. The first one is **Execution Coverage (EC)** wherein which the test generation is configured to achieve *block execution coverage*, a structural metric that indicates whether each block is executed during simulation. This coverage metric is most comparable to statement coverage in code coverage analysis. The second one is **Modified Condition Decision Coverage (MCDC**) which is a more rigorous coverage criterion. In the context of SDV, MCDC test generation checks for combinations of inputs that exercise each possible condition in the model, ensuring that every logical decision is fully tested. This is crucial for validating complex models, as it provides a higher level of assurance compared to EC alone.

—FT is a search-based method for generating tests violating system requirements or assumptions [1]. Formally, given an STL property ϕ of a model $\mathcal{M}$, FT aims to find a test case $\mathbf{t}$ s.t. $\rho(\phi, O(\mathbf{t}, \mathcal{M})) < 0$, i.e., $O(\mathbf{t}, \mathcal{M}) \not\models \phi$. In general, FT's objective is to produce a test that violates the property under examination, aiming to uncover bugs.

In our experiments, we employ the following three approaches for FT:

(1) FT-Ψ-TaLiRo: We use Ψ-TaLiRo, an open-source toolbox featured in the ARCH competition, designed for robustness-guided falsification of temporal logic properties in CPSs. It implements the PART-X algorithm (Partitioning with X-distributed sampling) [55], which dynamically divides the search space to locate the falsifying inputs. Specifically, the algorithm leverages local Gaussian process estimates to refine its search, adaptively partitioning and sampling the input space to focus on regions likely to contain falsifying points.
(2) FT-*S*-TaLiRo: We also employ the S-TaLiRo tool, an open-source FT toolbox for CPSs, featured in the ARCH competition, as outlined in [17]. This tool enhances the performance of falsification methods by applying coverage metrics to the state space of hybrid systems.
(3) FT-Confidence: We also apply the falsification algorithm that incorporates a robustness-based confidence metric, as described in [64]. The confidence measure (which is calculated by dividing the input space into finite subregions) is defined in terms of a coverage criterion of the input space that evaluates how thoroughly the entire input space is explored while focusing on local areas where low robustness values are detected.

For addressing *RQ1*, we consider the two FCT techniques (FCT-C and FCT-S) and the five state-of-the-art testing techniques: ART, OD-VB, OD-FB, SDV-EC and SDV-MCDC. Since FT is a property-directed test generation technique aimed at fault revelation rather than achieving coverage, we utilize it specifically for addressing RQ2 in our mutation testing experiments. Consequently, *RQ2* encompasses all the test generation strategies mentioned above. Note that we conducted 30 repetitions of each test generation algorithm to minimize the impact of random variations and provide statistical data. We conducted test generation experiments with the proposed FCT approaches (FCT-C and FCT-S), as well as ART, OD-based and SDV-based methods, on normal (non-faulty) models. In contrast, experiments with FT-based approaches were performed on faulty models.

4.1.2 Simulation Platform. We used MATLAB/Simulink R2018b from MathWorks for our experiments and the RTAMT library [53] for offline evaluation of STL properties. The experiments were executed on a MacBook Pro with macOS Ventura, Apple M1 chip, and 16 GB RAM.

4.1.3 Experimental Subjects. To answer our RQs, we considered five industrial, open-source, and publicly accessible Simulink models [21, 38–40] and ten requirements, stated as STL properties, that must be satisfied by these systems. These models have been widely used in other prior studies [7, 21, 52, 61].

Fault-Tolerant Fuel Control System (FCS). FCS is a benchmark from the automotive domain representing a fuel control system for a gasoline engine [39]. FCS is modeled as a hybrid system mathematically expressed using differential equations with a switching condition. FCS receives two inputs: throttle and engine speed. It has two outputs: fuel flow rate and the air-to-fuel ratio (λ). The control objective is to keep λ close to the reference (i.e., stoichiometric) mixture ratio (λ_{ref}).

Automatic Transmission Controller System (ATCS). ATCS models an automotive drivetrain in Simulink integrated with a Stateflow block to mimic the drivetrain dynamics [40]. The ATCS inputs are the values of the throttle and the brake. These can both assume any value in the interval $[0, 100]$ in each timestep. The ATCS outputs are the engine's speed ω (RPM), the *gear* number, and the car's velocity v (mph). ω and v are continuous state variables, while *gear* is discrete.

AECS. AECS from the avionics domain models a pair of aircraft elevators controlled by redundant actuators for a fault detection, isolation, and recovery application [38]. AECS has one input (Pilot Command) and two outputs (left elevator position and right elevator position). The goal of the control system is to keep the actuator position steady when the aircraft is flying at the designated altitude [8].

Neural-Network (NN) *Controller.* This NN controller is designed to maintain a magnet at a specified reference position above an electromagnet, ensuring it hovers steadily [21, 44]. The model accepts as input a reference position value (Ref), ranging from 1 to 3, and outputs the hovering magnet's true position (Pos). The objective is to ensure that following adjustments to the reference, the current position consistently converges towards that reference with the least amount of discrepancy.

Steam Condenser (SC) *with Recurrent NN Controller.* The model, as introduced in [67] and used also in [21], represents a dynamic SC model, regulated by a recurrent NN in feedback. The system's input can fluctuate within the range $[3.99, 4.01]$. The primary aim is to ensure that the output pressure steadily stabilizes within a pre-defined tolerance band.

Table 3 summarizes the key characteristics of these five models in terms of the number of blocks, number of lines, simulation time T (in seconds), sampling time (in seconds), and number of samples. Table 4 specifies their requirements in natural language and STL. It is important to note that most of the Simulink models (and their specifications) we used in our experiments are standard benchmarks widely used in the falsification community, particularly those from the well-known ARCH competition [20, 21, 48]. Also, in our experiments, we have ensured that the inputs generated by all the test generation solutions in our comparison follow the same shape and constraints.

For the AECS model, we use a square waveform for the Pilot Command input, characterized by its amplitude and frequency, as previously used in [5]. For the other experimental models (ATCS, FCS, NN, and SC), we follow the input shapes and specifications provided by the ARCH competition. To address diverse and complex input signal requirements, our approach supports various parameterization techniques with interpolation functions (e.g., piecewise constant, linear, and piecewise cubic) to generate the input values.

4.1.4 ***Property-Based Mutation Testing (PBMT)***. To perform the mutation testing experiments required to answer RQ2, we refer to PBMT [8]. Regular mutation testing requires generating a test that produces two different outputs when executed on the original and mutated programs,

Table 3. Key Features of Selected Simulink Models

Model	*#Blocks*	*#Lines/Connections*	T	Sample Time	*#Samples*
FCS	215	190	110	0.01	11,001
ATCS	65	92	30	0.04	751
AECS	825	577	10	0.01	1,001
NN	105	123	40	0.001	40,001
SC	173	144	35	0.01	3,501

Table 4. Properties of Experimental Subjects in Natural Language and STL

Property	**Ref.**	**Natural Language**	**STL**
Fault-Tolerant FCS			
ϕ_1^{FCS}	[52]	Always, within the interval $[\tau, T]$, the air-to-fuel ratio (λ) must not evolve beyond the tolerance bounds.	$\Box_{[\tau,T]}(0.8\lambda_{ref} \leq \lambda \leq 1.2\lambda_{ref})$
ϕ_2^{FCS}	[26]	The fuel flow rate should not be 0 for more than 1 second within the next 100 seconds period.	$\neg\Diamond_{[0,100]}\Box_{[0,1]}(\text{Fuel Flow Rate} = 0)$
ATCS			
ϕ_1^{ATCS}	[26]	"The engine speed never reaches $\bar{\omega}$".	$\Box(\omega < \bar{\omega})$
ϕ_2^{ATCS}	[26]	"The engine and the vehicle speed never reach $\bar{\omega}$ and $\bar{v}$, resp."	$\Box((v < \bar{v}) \wedge (\omega < \bar{\omega}))$
ϕ_3^{ATCS}	[26]	"If engine speed is always less than $\bar{\omega}$, then vehicle speed cannot exceed $\bar{v}$ in less than T seconds."	$\neg(\Diamond_{[0,T]}(v > \bar{v}) \wedge \Box(\omega < \bar{\omega}))$
ϕ_4^{ATCS}	[26]	"Within T seconds, the vehicle speed is above $\bar{v}$ and from that point on, the engine speed is always less than $\bar{\omega}$)."	$\Diamond_{[0,T]}((v \geq \bar{v}) \wedge \Box(\omega < \bar{\omega}))$
AECS			
ϕ_1^{AECS}	[5]	Whenever the pilot command, denoted as *cmd*, exceeds the threshold *m*, the measured actuator position, *pos*, must not exceed a distance of *n* units from *cmd* within a time frame of Q+a units.	$\Box(\uparrow(cmd \geq m) \rightarrow \Diamond_{[0,Q]}\Box_{[0,a]}(\lvert cmd - pos\rvert \leq n))$
NN Controller			
ϕ_1^{NN}	[21]	Within the time horizon of 40 time units, the output will be always close to the reference signal within 2 seconds.	$\Box_{[1,37]}\big(\lvert Pos - Ref\rvert > \alpha + \beta\lvert Ref\rvert \rightarrow \Diamond_{[0,2]}\Box_{[0,1]}\neg(\alpha + \beta\lvert Ref\rvert \leq \lvert Pos - Ref\rvert)\big)$
ϕ_2^{NN}	[21]	Given the reference signal Ref satisfying $1.95 \leq Ref \leq 2.05$, the output satisfies a conjunctive requirement defined over different time intervals.	$\Diamond_{[0,1]}(Pos > 3.2) \wedge \Diamond_{[1,1.5]}\big(\Box_{[0,0.5]}(1.75 < Pos < 2.25)\big) \wedge \Box_{[2,3]}(1.825 < Pos < 2.175)$
Steam Condenser with Recurrent Neural Network Controller (SC)			
ϕ_1^{SC}	[21]	Always, within the interval $[30, 35]$, the pressure never goes below 87 and beyond 87.5.	$\Box_{[30,35]}(87 \leq Pressure \leq 87.5)$

Parameters: $\lambda_{ref} = 14.6$, τ (settling time) = 10 seconds, $\bar{\omega} = 4{,}500$ RPM, $\bar{v} = 120$ mph, $P = 4$ seconds, $m = 0.09$, $Q = 2$ seconds, $a = 1$ second, $n = 0.02$, $\alpha = 0.005$, $\beta = 0.03$.

which is of little interest with Simulink models. It is trivial to generate a test that activates most of the components of a Simulink model due to the dataflow-oriented nature of the computations, and differences can be easily observed in the outputs. However, a test does an excellent job in exercising a fault only if the effect of the fault is amplified to the extent it causes a violation of a requirement (i.e., an STL property). Intuitively, a test that exercises a fault without causing the violation of a requirement is a test that does not fail when executed on the faulty model and thus cannot reveal the fault. PBMT addresses this issue, requiring the generation of tests that kill mutants by violating the available properties.

In particular, given a model $\mathcal{M}$, an STL property Φ of $\mathcal{M}$, and a test suite $\mathcal{T}$ consisting of a set of valid test cases for $\mathcal{M}$. A mutant $\mathcal{M}'$ of $\mathcal{M}$ is said to be *ϕ-killed* by $\mathcal{T}$ iff $\exists \mathbf{t} \in \mathcal{T}$ s.t.

Table 5. Mutation Operators and Number of Mutants

Type	Operator	# Mutants				
		FCS	ATCS	AECS	NN	SC
Line mutation	`Noise`	10	13	17	10	10
	`Bias/offset`	12	13	17	10	10
	`Negate`	10	13	17	0	0
Block mutation	`ROR`	0	0	10	0	0
	`S2P`	1	1	3	2	3
	`P2S`	0	2	6	1	2
	`ASR`	3	3	8	3	9
Total		36	45	78	26	34

$\rho(\Phi, O(\mathbf{t}, \mathcal{M})) > 0 \wedge \rho(\Phi, O(\mathbf{t}, \mathcal{M}')) < 0$. More simply, a test suite Φ-*kills* a mutant $\mathcal{M}'$ if and only if it has at least one test case that violates Φ on $\mathcal{M}'$ and satisfies Φ on $\mathcal{M}$.

A mutant $\mathcal{M}'$ is ϕ-*trivially different* [8] from $\mathcal{M}$ iff $\nexists \mathbf{t} : \rho(\Phi, O(\mathbf{t}, \mathcal{M})) > 0 \wedge \rho(\Phi, O(\mathbf{t}, \mathcal{M}')) < 0$. In simple words, there is no test case that can differentiate between $\mathcal{M}$ and $\mathcal{M}'$ w.r.t. Φ. For a Simulink model $\mathcal{M}$ with property Φ and a given set of mutation operators, the mutation score $\mathcal{MS}_\Phi$ achieved by a test suite $\mathcal{T}$ is given by

$$\mathcal{MS}_\Phi = \frac{\#\ \Phi\text{-}killed\ mutants}{\#\ non\ \Phi\text{-}trivially\ different\ mutants}$$

In our experiments, we use FIM [6], a fault injection tool for automatic fault injection and mutation of Simulink models. We use seven faults/mutation operators from FIM that are typically used in testing of Simulink models: three line mutations (`Noise`, `Bias/Offset`, and `Negate`) and four block mutations (**Relational Operator Replacement (`ROR`)**, **Sum to Product Mutation (`S2P`)**, **Product to Sum Mutation (`P2S`)**, and **Arithmetic Sign Replacement (`ASR`)**). We excluded FIM's `Absolute` operator due to its tendency to generate ϕ-trivially different mutants [8], and the `Invert` and `Bit-flip` operators due to their tendency to generate invalid mutants. Table 5 shows, for each subject, the number of valid mutants (i.e., mutants that could be compiled) generated for each mutation operator. We refer the reader to the work by Bartocci et al. [6] for a detailed description of these operators. For each valid first-order mutant[8] generated using FIM, we assume that the fault is active throughout the simulation, starting at time 0 and continuing until the end at time T.

For our mutation testing evaluations with PBMT, we exploit all the test generation solutions described in Section 4.1.1. Note that the set of selected techniques allow us to compare FCT-S (property-dependent) and FCT-C (property-independent) to other property-dependent (FT) and property-independent (ART, OD-VB, OD-FB, SDV-EC and SDV-MCDC) methods. Also, in our FT-based test generation setup for PBMT, FT is applied to each mutant individually. More formally, for every mutant $\mathcal{M}'$, FT tries to produce a property-violating test case $\mathbf{t}$ s.t. $O(\mathbf{t}, \mathcal{M}') \not\models \phi$. For the PBMT experiments, we simulate each mutant (i.e., faulty model) using the test suites generated by various test generation strategies to determine whether each test suite is effective in ϕ-killing the mutants.

Setup for FT. In our FT experiments with S-TaLiRo [17], Ψ-TaLiRo [55], and the work referenced in [64], we did not stop the test generation process after detecting the first fault. Instead, we continued

[8]In this work, we focus only on *first-order mutants* of Simulink models.

generating tests until the entire budget was exhausted. This ensured that the falsification tools explored more of the input space, similar to the other methods we evaluated. Specifically, for each mutant, we generated tests by finding counterexamples to the STL property, and the overall testing budget was divided equally among the total number of mutants.

4.2 Results and Discussion

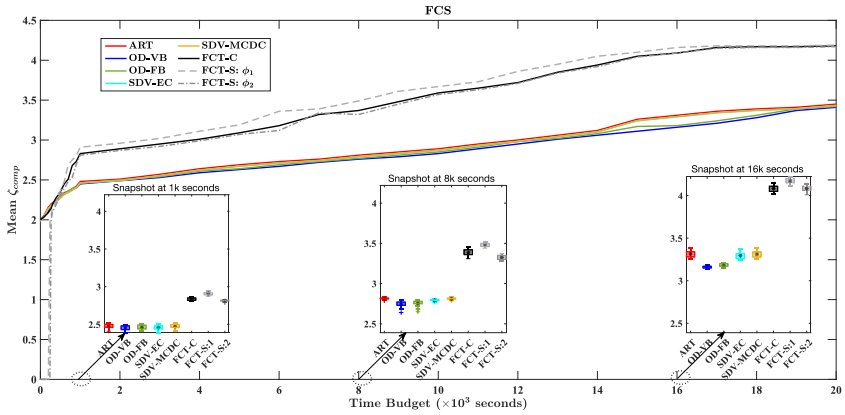
RQ1—Coverage. RQ1 evaluates the capability of state-of-the-art test generation strategies to thoroughly exercise features. As mentioned, the interconnected nature of the elements in dataflow models often lead to the execution of multiple elements when very few test cases are executed. This was also true in our experiments, where a single random test case was able to cover all the signals of the ATCS model, and an average of 9.25 random test cases were sufficient to cover >90% of the signals of the other four considered models. This evidence further stresses the need of more sophisticated coverage criteria. To this end, we study the signal feature coverage achieved using traditional test generation strategies with that of our proposed FCT-S and FCT-C techniques.

Figure 3 illustrates the trends of ζ_{comp} over time, averaged across 30 independent runs for each experimental subject and all properties. However, for a detailed view of the variability in the ζ_{comp} values for all the techniques, we refer the readers to Appendix A.1. Notably, ART, OD-VB, OD-FB, SDV-EC, SDV-MCDC and FCT-C are property-independent techniques, whereas FCT-S is a property-directed technique. Therefore, distinct curves are depicted for FCT-S corresponding to each property. Moreover, as FCT-S involves the identification of significant signals and features through the feature identification step, we notice that initially, the coverage with FCT-S remains at zero.

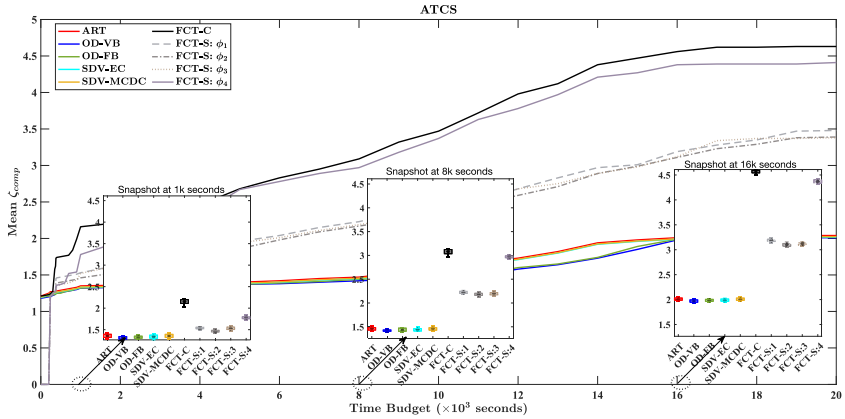
Upon analyzing the figure, we notice an initial short period during which both our proposed FCT-C and FCT-S techniques exhibit subpar performance in terms of the coverage metric. However, as time progresses, there is a noticeable and quick improvement, with our approaches eventually surpassing the state-of-the-art methods. This observation suggests that our techniques possess the capability to achieve high coverage, provided they are allocated more resources (allowed more time) to execute. Table 6 reports (1) the time required to identify significant signals and features for FCT-S technique: **Feature Identification Time (FIT)** and (2) the time for FCT-C/FCT-S surpassing state-of-the-art: **Surpass Time (ST)** w.r.t. ζ_{comp}. Based on the data presented in Table 6, we note that the feature identification step, on average, takes approximately 252.39 seconds across all experimental subjects. Moreover, we observe that FCT-C consistently outperforms FCT-S in terms of surpassing state-of-the-art strategies (i.e., *ST*), with FCT-C generally achieving higher values of ζ_{comp} than state-of-the-art techniques earlier than FCT-S.

Observing Figure 3, it becomes apparent that the curves representing the mean ζ_{comp} for various state-of-the-art strategies, namely ART, OD-VB, OD-FB, SDV-EC, and SDV-MCDC either overlap or closely align, suggesting comparable performance among these techniques. In particular, there is a considerable overlap between the curves for ART, SDV-EC and SDV-MCDC. Also, as anticipated, the Comprehensive (FCT-C) approach, designed to comprehensively cover all features and signals, consistently achieves the highest coverage levels compared to all other methods, including the Specification-directed (FCT-S) technique. Nevertheless, we note an interesting exception to this trend. Specifically, for the FCS model evaluated against property ϕ_1^{FCS}, the FCT-S strategy attains slightly higher coverage values than FCT-C. In this instance, we hypothesize that the superior mean ζ_{comp} achieved with FCT-S relative to FCT-C can be attributed to its ability to effectively cover key features of key signals, that in turn cause the extensive coverage of multiple features of multiple signals. This observation suggests that the FCT-S strategy may capture certain feature values that could be missed or challenging to cover with the Comprehensive (FCT-C) approach.

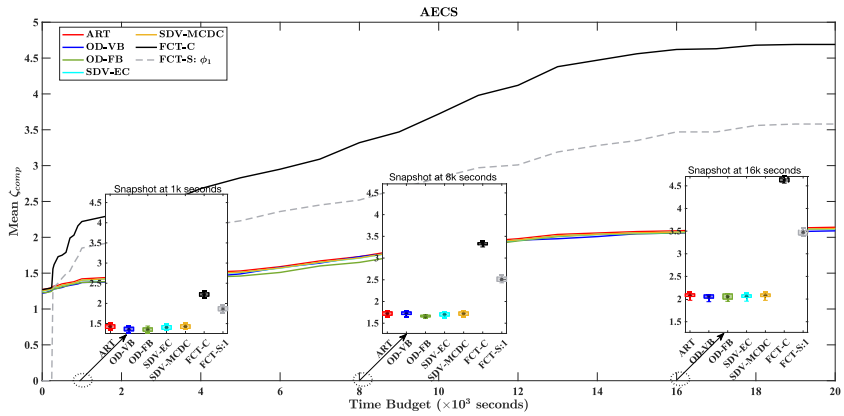
Referring to Figure 3, we note that both FCT-C and FCT-S reach saturation towards the later stages, typically around 16k–18k seconds, suggesting that further enhancement of coverage may become



(a) FCS model

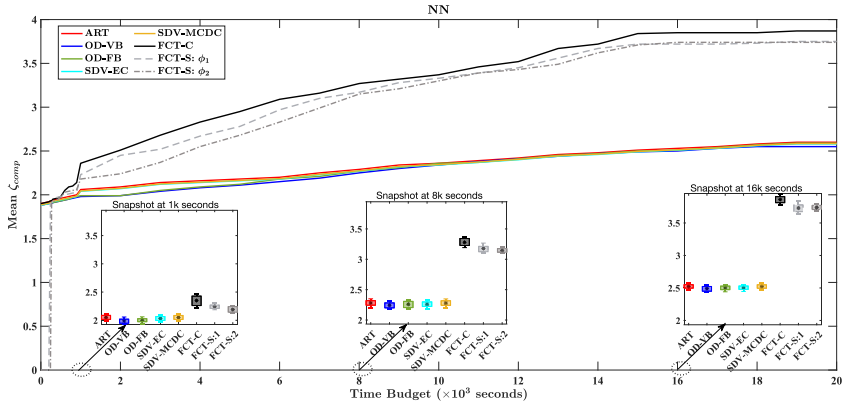


(b) ATCS model

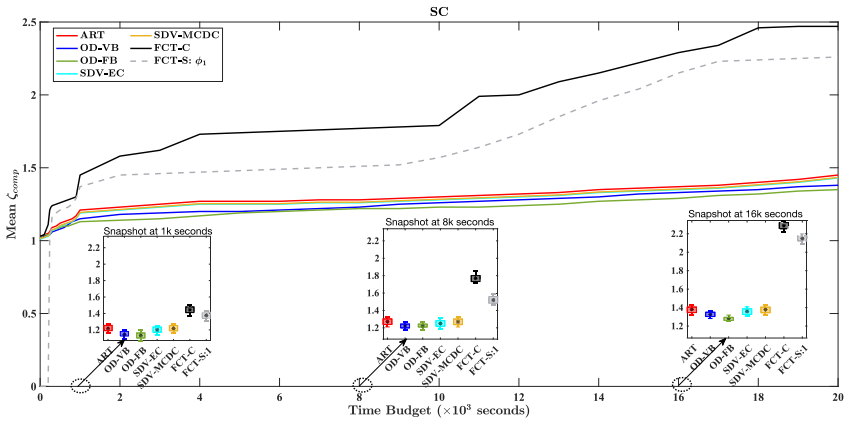


(c) AECS model

Fig. 3. Trends over time of mean value of ζ_{comp} for different test generation strategies for various models.



(d) NN model



(e) SC model

Fig. 3. Continued

Table 6. Time Required for Identifying Significant Signals and Features, and Times at Which FCT-S/FCT-C Surpass State-of-the-Art Strategies w.r.t. ζ_{comp}

Model	Property	FIT (s)		ST for FCT-S (s)		ST for FCT-C (s)	
		Average	Worst	Average	Worst	Average	Worst
FCS	ϕ_1	241.33	254.81	450.23	462.56	400.67	405.34
	ϕ_2	270.24	279.65	471.02	482.10		
ATCS	ϕ_1	243.67	278.67	380.23	397.24	352.67	365.38
	ϕ_2	246.73	269.66	384.56	395.26		
	ϕ_3	245.79	265.57	379.68	387.91		
	ϕ_4	244.83	271.58	372.59	380.26		
AECS	ϕ_1	272.80	293.72	406.27	411.84	274.56	281.67
NN	ϕ_1	246.52	270.43	403.86	415.27	304.53	311.25
	ϕ_2	271.48	291.27	502.36	516.28		
SC	ϕ_1	240.56	261.78	306.74	309.38	245.56	253.38

increasingly challenging. In contrast, state-of-the-art techniques do not exhibit full saturation in ζ_{comp} values, implying that there is still potential for improvement. This indicates that these techniques require additional (time) resources to generate tests that effectively cover the signal features.

In Figure 3, we also illustrate the variation in ζ_{comp} across 30 runs for each test generation strategy using box plots at different time snapshots. Each box in the plot comprises 30 data points, where each data point represents the ζ_{comp} achieved by a test suite generated in a single run. The horizontal axis depicts the test generation approach, while the vertical axis indicates the corresponding ζ_{comp} values. Within each box, an asterisk (*) mark indicates the mean value of ζ_{comp}. Referring to Figure 3(a), we observe minimal variability in the data for the FCS model, except for certain snapshots of time. For instance, at the 8k second snapshot, both OD-VB and OD-FB exhibit outliers (shown as lower whiskers in their respective boxes), suggesting that some test suites generated using these techniques achieve coverage levels lower than the mean value. Further, upon examining all the box plots of FCS model across various snapshots of time, we note that ART demonstrates slightly superior performance in terms of ζ_{comp} compared to traditional strategies. However, its performance lags behind when compared to our FCT techniques.

Compared to all other models, the test suites generated for the NN model (Figure 3(d)) exhibit slightly higher variations in the ζ_{comp} values. Additionally, for some models, particularly FCS (Figure 3(a)) and AECS (Figure 3(c)), across all snapshots of time, the box plots corresponding to FCT-C and FCT-S closely align at the same levels, indicating a close match in the coverage achieved by both techniques. This alignment suggests that FCT-C and FCT-S have similar effectiveness in achieving coverage levels for the given properties for these models.

In all the experimental subjects, the ART, OD-based, and SDV-based test generation approaches exhibit low coverage levels. In contrast, the FCT-C and FCT-S approaches involve targeted test generation aimed at maximizing feature coverage. As a result, the generated tests extensively exercise the target features of the target internal signals, leading to substantially higher coverage levels compared to traditional test generation solutions, as illustrated in Figure 3. These results highlight the existence of a gap in the capability of baseline techniques to cover signal features in CPS Simulink models.

It is important to note that the signals and features identified through our feature identification process may not necessarily correspond to the *true* signals and features influencing the STL property. There could be discrepancies between the identified signals and features, and those that truly influence the satisfaction of the considered property, namely the **Ground Truth (GT)** signals and features. Furthermore, there is no definitive method for identifying the GT signals/features, as it is up to the tester to decide or select a specific procedure for this purpose. To study how our feature identification step performs in the identification of these features, we employ correlation analysis over a large set of tests among internal signals, features, and the quantitative robustness of simulation traces w.r.t. the STL properties, to determine the GT signals and features. This analysis is conducted over an extended period, involving 10,000 random tests for each model and property, for a total of approximately 74 hours for all our experimental subjects and properties.

Once we computed the GT signals and features, we assessed the extent to which the generated test suites cover the *GT* signals and features. To this end, we evaluate the *ϕ-driven signal feature coverage* of the GT signals and their associated features. Figure 4 illustrates the trends of ζ_ϕ, representing the *ϕ-driven signal feature coverage* computed for the GT signals and features, averaged over 30 runs. We also show the variability in ζ_ϕ across all models and properties in the Appendix A.2. It is important to mention that we have one plot for each property of a model because GT features and signals are specific to each property. Also, it is important to highlight that the FCT-S-based test suites we utilize to compute ζ_ϕ-values for GT features and signals, are the ones that we

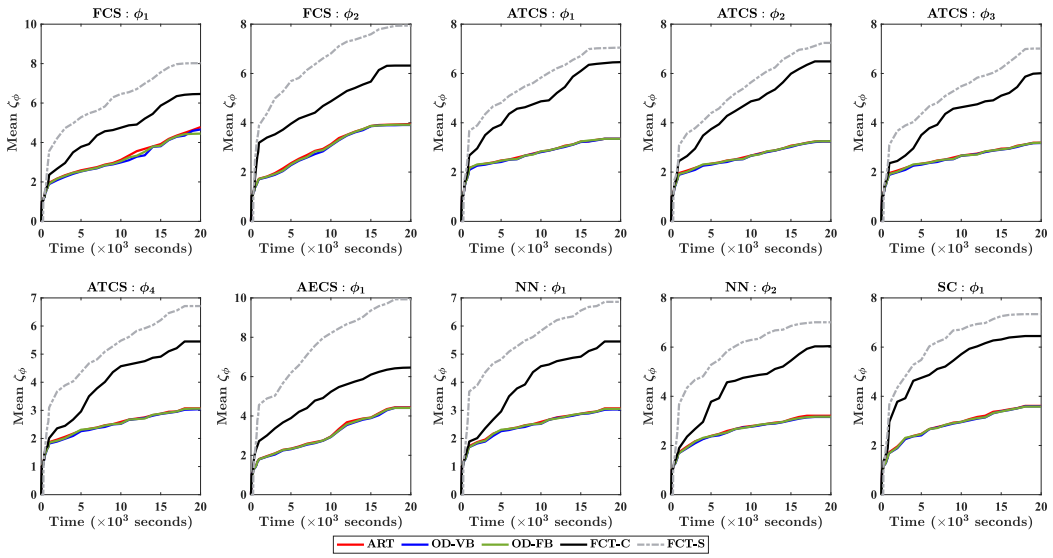


Fig. 4. Trends over time of the mean ζ_ϕ-values computed for GT signals and features.

generated focusing on maximization of coverage of signals and features identified through the feature identification algorithm. This approach provides valuable insights into the effectiveness of our time-efficient feature identification strategy and offers indications of any correlation between the *identified signals+features* and the *GT signals+features*.

Upon examination of Figure 4, distinct patterns emerge, delineating three main categories within the curves. First, one set of curves indicates that the state-of-the-art testing approaches inadequately exercise the GT features and signals, that is, they fail to extensively exercise the signals that matter most for the property under test. Second, another set of curves suggests that FCT-C demonstrates better coverage of these features and signals compared to the state-of-the-art methods, that is, specific methods are needed to extensively cover relevant features of relevant signals. Lastly, a third set of curves highlights that FCT-S surpasses both FCT-C and the state-of-the-art, achieving higher coverage of the GT features and signals. This behavior holds for all the experimental models and properties. This observation demonstrates the effectiveness of our feature identification step in identifying features and signals that contribute to the coverage of GT signals and features, implying a degree of correlation between them. From Figure 4, we also observe that across all the experimental models and properties, there are huge overlaps among the curves corresponding to the state-of-the-art testing strategies, indicating their comparable performance in exercising the GT signal features.

In addition, upon observing Figure A1 (Appendix A.1) regarding the variability in ζ_{comp} values, it is evident that all the tubes are closely clustered, indicating minimal dispersion between the minimum and maximum values. Additionally, there are noticeable overlaps among the tubes corresponding to the state-of-the-art techniques across most of our experimental subject models and properties. A similar trend is observed in Figure A2 (Appendix A.2) for the variability in the trends of GT signal feature coverage ζ_ϕ. This visualization employs the same tubular format, capturing minimum, maximum, and mean ζ_ϕ values. Notably, the tightly packed tubes suggest limited variance in the data. Due to the significant overlap among the data points for ART- and SDV-based test suites, we have excluded the SDV-EC and SDV-MCDC curves from Figures 4, A1, and A2 to enhance the readability of the plots. In both Figures A1 and A2, a distinct contrast in performance

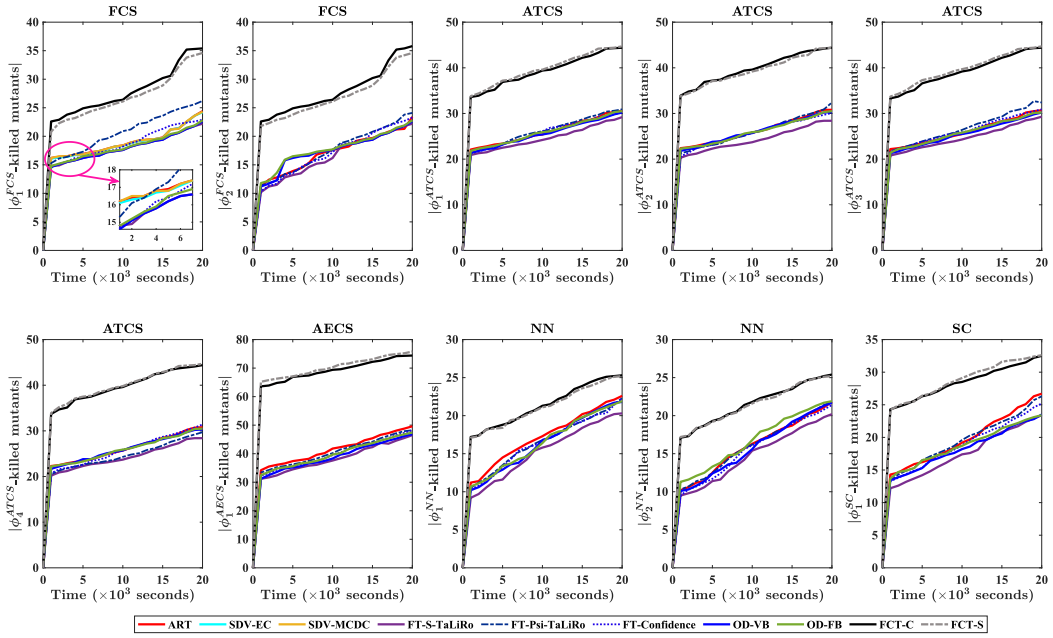


Fig. 5. Trends over time of the average number of ϕ-killed mutants by different test generation strategies.

is observed between our feature-coverage-based testing techniques and the state-of-the-art methods, with the feature-coverage-based testing techniques showcasing superior performance.

Summary for RQ1

State-of-the-art test generation solutions do not effectively cover the signal features of CPS dataflow models, neither considering the full set of features and signals nor considering the signals and features relevant to the tested property. Our Feature-Coverage Testing approaches (both Comprehensive and Specification-directed) achieve better performance (i.e. higher coverage levels) than the state-of-the-art methods. Specification-directed Feature-Coverage test generation can exercise the signals and features relevant to property more thoroughly than Comprehensive Feature-coverage test generation.

RQ2—Fault-Revealing Capability. RQ2 investigates if test suites generated according to signal feature coverage have a better fault-revealing capability than the test suites generated with state-of-the-art methods. In particular, we compare the mutant-killing capability of the test suites generated by ART, SDV-EC, SDV-MCDC, FT-S-TaLiRo, FT-Ψ-TaLiRo, FT-Confidence, OD-VB and OD-FB to the test suites generated with FCT, namely by FCT-C and FCT-S.

Figure 5 illustrates the trends over time of the number of mutants ϕ-killed by the test suites generated using various test generation methods, according to the same setup used for RQ1. For better visualization and ease of interpretation, the plots show the average number of mutants ϕ-killed. However, we refer the readers to the Appendix B for a comprehensive view of the variability in the number of mutants ϕ-killed according to PBMT.

We verified the ϕ-killability of the mutants exploiting the mutant directed test generation strategy illustrated in [8]. All the generated mutants were found to be ϕ-killable. As evident from Figure 5,

distinct patterns emerge among the curves. One set of curves, represented by FCT-C and FCT-S, consistently exhibit a notably higher number of ϕ-killed mutants compared to all other state-of-the-art testing methods. This trend holds across all models and properties, giving strong evidence of the benefit of generating test cases that thoroughly exercise the features of the internal signals.

An intriguing observation is that FT, which aims to falsify the property, does not achieve a higher mutant kill count, and always performed less effectively compared to FCT-S and FCT-C. This trend remains consistent for all FT-based methods (FT-S-TaLiRo, FT-Ψ-TaLiRo, FT-Confidence) across all system models and their respective properties, indicating that the approach of targeting falsification was less effective in identifying faults (i.e., killing mutants according to the definition of ϕ-killed mutants of PBMT) compared to the strategies employed by FCT-S and FCT-C. However, given the scope of our study, further investigation would be needed to fully assess the relative effectiveness of falsification in identifying faults compared to the strategies employed by FCT-S and FCT-C. Likewise, the ART-based test generation, which prioritizes diversity in test cases without considering the presence of a property, demonstrates degraded performance in terms of mutant-killing. It is not surprising that the test cases generated with ART are insufficient to adequately (i) explore the internal behavior of the software and (ii) ensure the detection of even basic faults (such as those we seeded using FIM) that could impact the property. This finding aligns with previous literature [8], further corroborating the inadequacy of ART-generated test cases for effectively identifying faults.

Also, the performance of the SDV-based test suites in ϕ-killing the mutants is found to be quite comparable to that of the ART-based test suites, following similar trends in signal feature coverage. While there is no definitive reason for this comparable performance, we conjecture that ART, though more randomized, generates a large number of diverse test cases. In contrast, SDV-based suites are more systematic and structured, but their exhaustive nature allows them to cover much of the same ground as ART, particularly for structural coverage metrics like EC and MCDC. This overlap ensures that, despite their different approaches, both methods achieve similar signal feature coverage and are equally effective at identifying faults (mutants) in the system models.

Moreover, the OD-based testing strategies (OD-VB and OD-FB) exhibit limited fault-revealing capabilities. The rationale behind employing OD-testing strategies is to enhance the likelihood of detecting failures by diversifying test output signals, thereby increasing the chances of identifying significant discrepancies between expected and actual signals. However, based on our PBMT evaluations across all models and properties, we conjecture that the poor mutant-killing performance of OD-VB and OD-FB stems from inadequate exploration of the feature value landscape of internal signals. More specifically, while the OD-FB approach prioritizes diversity in the feature vectors of output signals, it overlooks the diversification of features within internal signals. We hypothesize that this limitation arises due to the intricate interconnection of components and the complex interplay between signals within the system. In a nutshell, OD(-VB and -FB) testing fails to comprehensively investigate the internal behavior of the system model.

On the contrary, FCT-S and FCT-C generate test cases that achieve higher feature coverage and are more effective in detecting and capturing errors in the system, leading to higher mutant-killing capability. These results show how generating tests that cover a range of features, and a range of feature values, is extremely beneficial to obtain tests that can actually reveal faults, exercising them in such a way that they observably impact on the safety properties. In fact, by targeting the features of internal signals, FCT increases the likelihood of activating faults or erroneous behavior associated with those features. This is because faults in CPS are often related to specific signal characteristics or interactions between signals.

In a nutshell, FCT-S and FCT-C revealed to be the most effective test case generation strategies, killing the highest number of mutants. Evidently, the tests generated with FCT-S and FCT-C exercise

Table 7. Summary of Results of PBMT for Individual Mutation Operators

	Noise	Bias/Offset	Negate	ROR	S2P	P2S	ASR
# Mutants generated	60	62	40	10	10	11	26
# (%) of ϕ-trivially different mutants	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
# (%) non-ϕ-trivially different mutants NTD_ϕ	60 (100%)	62 (100%)	40 (100%)	10 (100%)	10 (100%)	11 (100%)	26 (100%)
# (%) NTD_ϕ *not killed* by ART+FT+OD+SDV	12 (20%)	11 (17.74%)	10 (25%)	10 (100%)	4 (40%)	3 (27.27%)	7 (26.92%)

the mutant in a way that alters the features of the internal signals, ultimately propagating the failure to the observable interface leading to the violation of the system property. Additionally, the higher effectiveness of FCT-C and FCT-S demonstrates how diversity in the feature values exercised by the test cases leads to a broader exploration of the system's response space and improves the chances of revealing different types of faults or corner cases that can be easily missed by feature-insensitive testing approaches.

Observing Figure B1 (Appendix B), we note that test suites produced using FCT-S and FCT-C demonstrate the ability to kill the highest number of mutants, achieving a maximum mutation score of 100% (i.e., killing all the ϕ-killable mutants) at around 18k seconds (i.e., 5 hours) for the majority of the experimental subjects. Additionally, we notice significant overlap among the tubes representing the state-of-the-art testing strategies for certain experimental subjects. Specifically, for AECS, NN (property 1), and SC models, the maximum number of mutants killed appears slightly higher for ART-based and SDV-based test suites. Nonetheless, it is essential to highlight that the overall mutant killing count for ART-based and SDV-based test suites remains substantially low compared to even the minimum number of mutants killed by our proposed FCT-C and FCT-S techniques.

Another observation is that surprisingly the FCT-S strategy performs comparably to the baseline FCT-C method. For most of our experimental models and properties, the disparities in the mean number of mutants ϕ-killed by FCT-C and FCT-S are minimal. The comparable performance of FCT-S and FCT-C-based testing methods suggests that the Specification-directed approach, although exercising the signals and features that strongly correlate with the tested property more thoroughly than the feature-comprehensive strategy, may not necessarily enhance mutant killing.

We also examined the correlation or patterns behind the failure of state-of-the-art techniques for individual mutation operators. Table 7 summarizes the results of the PBMT experiments for each mutation operator, including: (1) the number of mutants generated, (2) the number (and percentage) of ϕ-trivially different mutants, (3) the number (and percentage) of (non-ϕ-trivially different mutants NTD_ϕ, and (4) the number (and percentage) of NTD_ϕ mutants not killed by any of the state-of-the-art test generation techniques (including ART, FT-based, SDV-based, and OD-based approaches). Table 7 presents the combined results for all five of our experimental subjects. Note that the results we present are based on the PBMT experiments using the test suites obtained at a 20k-second snapshot. Although we did not identify a clear pattern for why the state-of-the-art techniques failed to handle specific mutation operators across our subject systems, we observed that all the mutants generated by the ROR operator were particularly difficult to ϕ-kill using any of the state-of-the-art testing methods, including ART, SDV, FT (FT-S-TaLiRo, FT-Ψ-TaLiRo, and FT-Confidence), and OD. However, our proposed FCT-based approaches successfully killed all the ROR mutants.

We finally examined the correlation between the coverage values of individual signal feature types and mutation scores to determine if specific features significantly contribute to mutation-killing capabilities. However, we did not observe a consistent pattern across all our experimental subjects regarding the influence of individual signal features on mutation-killing effectiveness. In our future work, we intend to delve deeper into the role of signal features in detecting faults.

Summary for RQ2

When testing CPS dataflow models w.r.t. formal specifications, our *Comprehensive* (FCT-C) and *Specification-directed* (FCT-S) feature-coverage testing methods outperformed state-of-the-art methods, yielding higher mutation scores. FCT-C and FCT-S performed comparably well, suggesting that specification-directed methods may not be strongly necessary to test CPS models.

4.3 Threats to Validity

The following threats to validity affect our evaluation.

External Validity. Like many other evaluations, our study faces an external validity concern regarding the generalizability of the results. To mitigate this concern, we experimented with multiple properties and CPS models across distinct application domains. The collected evidence is consistent across all properties and subject applications, suggesting our results represent a relevant and motivating starting point to investigate the notion of signal feature coverage further. The representativeness of the injected faults poses another threat. The reported results rely on typical faults and mutation operators tailored for Simulink models, aiming to provide a fair and relevant assessment with hundreds of injected faults. Designing a first study of similar size with real Simulink faults would be infeasible due to the need for a large dataset of faults.

Internal Validity. An internal validity concern pertains to the generated mutants. We considered first-order mutants to align with the prevalent research on mutation testing and evidence that suggests that they are good predictors of test suite effectiveness [29]. We left the investigation of multi-fault Simulink models to future research.

Conclusion Validity. One potential conclusion validity concern relates to the randomness of the experimented test generation methods. We addressed this concern repeating each test case generation process 30 times, reporting both average results (Figures 3, 4, and 5) and results about the variance of the computed metrics (Appendices A and B). The consistency of the evidence across all subjects and the limited variance of the results indicate that randomness is not a significant threat to our conclusions.

5 Related Work

There has been growing interest in coverage-based testing approaches for CPS dataflow models in recent years. We first mention the *input coverage* metric, which aims at achieving diversity among the input signals [34]. The *output coverage* aims at achieving diversity in system outputs [46]. The works in [34, 45, 46] exploit a method for generating tests that maximizes output signal diversity in Simulink models by providing guidance in terms of the output values that should be exercised. The rationale behind this approach is that diversifying the test output signals increases the likelihood of identifying significant differences between expected and actual signals, enabling engineers to detect failures more effectively. *Structural coverage* metrics for CPS dataflow models include block EC, decision coverage, condition decision coverage, MCDC, and node/edge/path coverage [2, 43, 65]. Though the traditional coverage criteria focus on input values, output responses, and structural elements, they do not fully capture the complexities and behaviors of CPS dataflow models. In particular, the dynamics of CPS dataflow models are influenced by internal signal features and interactions. Traditional testing strategies guided by input/output values do not guarantee that the internal behavior of the system is well exercised, missing to reveal faults, as reported in our

empirical validation. Signal feature coverage addresses these limitations by explicitly considering the features of internal signals within CPS dataflow models that effectively reveal system faults.

Others also explored the notion of signal features to enhance the testing of CPS models. For instance, Zander [70] introduced a comprehensive taxonomy of signal features tailored for dataflow models and advocated for their use in describing specifications and test oracles. Matinnejad et al. [46] similarly delved into signal features and emphasized the significance of diversifying signal features to enhance test generation processes. Others advocate that internal signals can be useful for fault identification and localization [24, 34]. In this article, we contribute to this body of knowledge by defining a novel way of exploiting features based on the notion of *Signal feature coverage* that drives test generation toward exercising the *features* of *internal* signals of CPS dataflow models. While our approach focuses on statistical and spectral signal features (detailed in Section 3.1), the taxonomy in [12] primarily addresses signal behaviors such as spikes, oscillatory behavior, and transient characteristics (e.g., rise time, fall time, overshoot, and undershoot). Both sets of features are complementary, with our work concentrating on quantifiable statistical and spectral attributes, whereas [12] explores specific behavioral patterns. We believe that combining both perspectives could lead to an even more comprehensive understanding of signal behavior in CPS dataflow models.

Another area of research considers the fusion of formal methods [49] and model checking techniques [51] with coverage-based testing. These approaches use formal specifications to guide the test case generation process and verify desired system properties [16, 19, 37]. Another work in [4] introduces an adaptive test generation strategy to achieve specification coverage for temporal logic properties. We also mention the work in [17] which leverages coverage metrics on the state space of hybrid systems to enhance the effectiveness of falsification methods. In contrast, our approach capitalizes on formal specifications (i.e., system safety properties expressed in STL) to identify critical signals and features, which are then taken into consideration for generating test cases based on signal feature coverage. Furthermore, researchers have explored the automation of test case generation using techniques such as genetic algorithms [3, 54] and search-based testing [66, 71]. Similar to these works that utilize search-based strategies for test case generation w.r.t. a given test objective, we employ PSO to generate test cases that maximize the coverage of signal features.

We also mention research on other coverage-based falsification methods that target the input or output space. For example, the work in [64] introduces a robustness-based confidence measure designed to evaluate the likelihood that a system cannot be falsified. This measure is based on a coverage criterion of the input space, which evaluates how thoroughly the entire input space has been explored, along with a focused exploration of regions where low robustness is detected. Additionally, we discuss the PART-X falsification algorithm (utilized in the Ψ-TaLiRo tool suite) [55], which adaptively partitions the search space around falsifying points. This algorithm employs local Gaussian process estimates to iteratively branch and sample within the input space. In our work, we also evaluate our proposed testing approach against these coverage-based falsification methods (FT-Ψ-TaLiRo, FT-S-TaLiRo and FT-Confidence) and demonstrate its superior effectiveness compared to current state-of-the-art techniques.

In addition, researchers have explored mutation-based testing of CPS Simulink dataflow models, where the focus is on detecting the seeded faults [13, 25]. Recently, PBMT [8] has been proposed to address mutation testing of CPS Simulink models w.r.t. STL properties. In our experiments, we leveraged this concept to demonstrate the mutant-killing capability of the test suite generated using our proposed FCT technique.

6 Conclusion and Future Work

In this article, we introduced the notion of signal feature coverage as a coverage metric based on common signal features and tailored for CPS dataflow models. We also designed a test generation strategy aiming to maximize the signal feature coverage for a given set of features and signals. We demonstrated the effectiveness of our approach against four state-of-the-art methods. We showed that by generating tests that improve the coverage of features of internal signals of a system model, the testing process has improved fault detection than state-of-the-art testing strategies. The improvement is observed consistently across all properties and subjects used in the experiments. From the results obtained, we extract the following key insights: (1) By leveraging the signal features, the resulting coverage criteria are better tailored to capture the behavior and nuances of the CPS models. (2) Test suites focusing on maximizing feature coverage of signals result in higher mutation killing score.

Our approach is agnostic to the source of the signals; as long as we can instrument and execute the system model, whether in a simulated environment or a real-world application, we can effectively apply our method. This flexibility ensures that our approach is not limited by the test environment, and it can be extended to practical scenarios provided that the system can be instrumented to capture the necessary signal data. While the current evaluation focuses on simulated environments, which are the norm for testing CPS dataflow models, future work could explore the application of our approach in real-world settings to further validate its practical performance.

In the future, we aim to extend our research to investigate the applicability of signal feature coverage to higher-order mutants, exploring its potential in more complex scenarios. Additionally, we plan to delve into the interplay of different signal features affecting system specifications to refine our testing approach further. These efforts promise to advance the knowledge in CPS testing, paving the way for more robust and reliable CPS in the future.

References

[1] Arvind S. Adimoolam, Thao Dang, Alexandre Donzé, James Kapinski, and Xiaoqing Jin. 2017. Classification and coverage-based falsification for embedded control systems. In *Proceedings of the 29th International Conference on Computer Aided Verification (CAV '17)*. Rupak Majumdar and Viktor Kuncak (Eds.), Lecture Notes in Computer Science, Vol. 10426, Springer, Cham, 483–503. DOI: https://doi.org/10.1007/978-3-319-63387-9_24

[2] Vadim Alyokhin, Benedikte Elbel, Martin Rothfelder, and Alexander Pretschner. 2004. Coverage metrics for continuous function charts. In *Proceedings of the 15th International Symposium on Software Reliability Engineering (ISSRE '04)*. IEEE Computer Society, USA, 257–268. DOI: https://doi.org/10.1109/ISSRE.2004.15

[3] Aitor Arrieta, Shuai Wang, Urtzi Markiegi, Goiuria Sagardui, and Leire Etxeberria. 2018. Employing multi-objective search to enhance reactive test case generation and prioritization for testing industrial cyber-physical systems. *IEEE Trans. Ind. Inf.* 14, 3 (2018), 1055–1066. DOI: https://doi.org/10.1109/TII.2017.2788019

[4] Ezio Bartocci, Roderick Bloem, Benedikt Maderbacher, Niveditha Manjunath, and Dejan Ničković. 2022. Adaptive testing for specification coverage and refinement in CPS models. *Nonlinear Anal.: Hybrid Syst.* 46 (2022), 101254.

[5] Ezio Bartocci, Niveditha Manjunath, Leonardo Mariani, Cristinel Mateis, and Dejan Nickovic. 2021. CPSDebug: Automatic failure explanation in CPS models. *Int. J. Softw. Tools Technol. Transf.* 23, 5 (2021), 783–796. DOI: https://doi.org/10.1007/S10009-020-00599-4

[6] Ezio Bartocci, Leonardo Mariani, Dejan Nickovic, and Drishti Yadav. 2022. FIM: Fault injection and mutation for simulink. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '22)*. Abhik Roychoudhury, Cristian Cadar, and Miryung Kim (Eds.). ACM, 1716–1720. DOI: https://doi.org/10.1145/3540250.3558932

[7] Ezio Bartocci, Leonardo Mariani, Dejan Nickovic, and Drishti Yadav. 2022. Search-based testing for accurate fault localization in CPS. In *Proceedings of the IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE '22)*. IEEE, 145–156. DOI: https://doi.org/10.1109/ISSRE55969.2022.00024

[8] Ezio Bartocci, Leonardo Mariani, Dejan Nickovic, and Drishti Yadav. 2023. Property-based mutation testing. In *Proceedings of the IEEE Conference on Software Testing, Verification and Validation (ICST '23)*. IEEE, 222–233. DOI: https://doi.org/10.1109/ICST57152.2023.00029

[9] Seyede Marziyeh Ghoreshi Beyrami and Peyvand Ghaderyan. 2020. A robust, cost-effective and non-invasive computer-aided method for diagnosis three types of neurodegenerative diseases with gait signal analysis. *Measurement* 156 (2020), 107579.

[10] Boualem Boashash. 1992. Estimating and interpreting the instantaneous frequency of a signal. I. Fundamentals. *Proc. IEEE* 80, 4 (1992), 520–538.

[11] Boualem Boashash. 1992. Estimating and interpreting the instantaneous frequency of a signal. II. Algorithms and applications. *Proc. IEEE* 80, 4 (1992), 540–568.

[12] Chaima Boufaied, Maris Jukss, Domenico Bianculli, Lionel Claude Briand, and Yago Isasi Parache. 2021. Signal-based properties of cyber-physical systems: Taxonomy and logic-based characterization. *J. Syst. Softw.* 174 (2021), 110881. DOI: https://doi.org/10.1016/J.JSS.2020.110881

[13] Angelo Brillout, Nannan He, Michele Mazzucchi, Daniel Kroening, Mitra Purandare, Philipp Rümmer, and Georg Weissenbacher. 2009. Mutation-based test case generation for simulink models. In *Proceedings of the 8th International Symposium on Formal Methods for Components and Objects (FMCO '09)*. Frank S. de Boer, Marcello M. Bonsangue, Stefan Hallerstede, and Michael Leuschel (Eds.), Lecture Notes in Computer Science, Vol. 6286, Springer, Berlin, 208–227. DOI: https://doi.org/10.1007/978-3-642-17071-3_11

[14] Tsong Yueh Chen, Fei-Ching Kuo, Huai Liu, and W. Eric Wong. 2008. Does adaptive random testing deliver a higher confidence than random testing? In *Proceedings of the 8th International Conference on Quality Software (QSIC '08)*. Hong Zhu (Ed.), IEEE Computer Society, 145–154. DOI: https://doi.org/10.1109/QSIC.2008.23

[15] Tsong Yueh Chen, Hing Leung, and Ieng Kei Mak. 2005. Adaptive random testing. In *Advances in Computer Science-ASIAN 2004: Proceedings of the 9th Asian Computing Science Conference on Higher-Level Decision Making*. Dedicated to Jean-Louis Lassez on the Occasion of His 5th Birthday. Springer, Berlin, 320–329.

[16] Adel Dokhanchi, Shakiba Yaghoubi, Bardh Hoxha, and Georgios Fainekos. 2017. Vacuity aware falsification for MTL request-response specifications. In *Proceedings of the 13th IEEE Conference on Automation Science and Engineering (CASE '17)*. IEEE, 1332–1337. DOI: https://doi.org/10.1109/COASE.2017.8256286

[17] Adel Dokhanchi, Aditya Zutshi, Rahul T. Sriniva, Sriram Sankaranarayanan, and Georgios Fainekos. 2015. Requirements driven falsification with coverage metrics. In *Proceedings of the 2015 International Conference on Embedded Software (EMSOFT '15)*. Alain Girault and Nan Guan (Eds.), IEEE, 31–40. DOI: https://doi.org/10.1109/EMSOFT.2015.7318257

[18] Alexandre Donzé and Oded Maler. 2010. Robust satisfaction of temporal logic over real-valued signals. In *Proceedings of the 8th International Conference on Formal Modeling and Analysis of Timed Systems* (FORMATS '10). Krishnendu Chatterjee and Thomas A. Henzinger (Eds.), Lecture Notes in Computer Science, Vol. 6246, Springer, Berlin, 92–106. DOI: https://doi.org/10.1007/978-3-642-15297-9_9

[19] Tommaso Dreossi, Thao Dang, Alexandre Donzé, James Kapinski, Xiaoqing Jin, and Jyotirmoy V. Deshmukh. 2015. Efficient guiding strategies for testing of temporal properties of hybrid systems. In *Proceedings of the 7th International Symposium on NASA Formal Methods (NFM '15)*. Klaus Havelund, Gerard J. Holzmann, and Rajeev Joshi (Eds.), Lecture Notes in Computer Science, Vol. 9058, Springer, Cham, 127–142. DOI: https://doi.org/10.1007/978-3-319-17524-9_10

[20] Gidon Ernst, Paolo Arcaini, Ismail Bennani, Aniruddh Chandratre, Alexandre Donzé, Georgios Fainekos, Goran Frehse, Khouloud Gaaloul, Jun Inoue, Tanmay Khandait, et al. 2021. ARCH-COMP 2021 category report: Falsification with validation of results. In *Proceedings of the 8th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH '21)*. Goran Frehse and Matthias Althoff (Eds.), EPiC Series in Computing, Vol. 80, EasyChair, 133–152. DOI: https://doi.org/10.29007/xwl1

[21] Gidon Ernst, Paolo Arcaini, Georgios Fainekos, Federico Formica, Jun Inoue, Tanmay Khandait, Mohammad Mahdi Mahboob, Claudio Menghi, Giulia Pedrielli, Masaki Waga, et al. 2022. ARCH-COMP 2022 category report: Falsification with ubounded resources. *EPiC Series in Computing* 90 (2022), 204–221.

[22] Georgios E. Fainekos and George J. Pappas. 2009. Robustness of temporal logic specifications for continuous-time signals. *Theor. Comput. Sci.* 410, 42 (2009), 4262–4291. DOI: https://doi.org/10.1016/J.TCS.2009.06.021

[23] Gordon Fraser and Andrea Arcuri. 2011. EvoSuite: Automatic test suite generation for object-oriented software. In *SIGSOFT/FSE '11: Proceedings of the 19th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-19) and ESEC '11: Proceedings of the 13th European Software Engineering Conference (ESEC-13)*. Tibor Gyimóthy and Andreas Zeller (Eds.). ACM, 416–419. DOI: https://doi.org/10.1145/2025113.2025179

[24] Gregory Gay, Ajitha Rajan, Matt Staats, Michael W. Whalen, and Mats Per Erik Heimdahl. 2016. The effect of program and model structure on the effectiveness of MC/DC test adequacy coverage. *ACM Trans. Softw. Eng. Methodol.* 25, 3 (2016), 25:1–25:34. DOI: https://doi.org/10.1145/2934672

[25] Le Thi My Hanh, Thanh Binh Nguyen, and Khuat Thanh Tung. 2014. Applying the meta-heuristic algorithms for mutation-based test data generation for simulink models. In *Proceedings of the 5th Symposium on Information and Communication Technology (SoICT '14)*. Nguyen Trong Giang, Huynh Quyet Thang, Ismal Khalil, Son Hong Ngo, Yves Deville, and Marc Bui (Eds.), ACM, 102–109. DOI: https://doi.org/10.1145/2676585.2676617

[26] Bardh Hoxha, Houssam Abbas, and Georgios Fainekos. 2014. Benchmarks for temporal logic requirements for automotive systems. In *Proceedings of the 1st and 2nd International Workshop on Applied veRification for Continuous and Hybrid Systems (ARCH@CPSWeek '14/ARCH@CPSWeek '15)*. Goran Frehse and Matthias Althoff (Eds.), EPiC Series in Computing, Vol. 34, EasyChair, USA, 25–30. DOI: https://doi.org/10.29007/xwrs

[27] Laura Inozemtseva and Reid Holmes. 2014. Coverage is not strongly correlated with test suite effectiveness. In *Proceedings of the 36th International Conference on Software Engineering (ICSE '14)*. Pankaj Jalote, Lionel C. Briand, and André van der Hoek (Eds.). ACM, New York, NY, 435–445. DOI: https://doi.org/10.1145/2568225.2568271

[28] Marko Ivankovic, Goran Petrovic, René Just, and Gordon Fraser. 2019. Code coverage at Google. In *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/SIGSOFT FSE '19)*. Marlon Dumas, Dietmar Pfahl, Sven Apel, and Alessandra Russo (Eds.), ACM, 955–963. DOI: https://doi.org/10.1145/3338906.3340459

[29] René Just, Darioush Jalali, Laura Inozemtseva, Michael D. Ernst, Reid Holmes, and Gordon Fraser. 2014. Are mutants a valid substitute for real faults in software testing? In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE '14)*. ACM, New York, NY, 654–665. DOI: https://doi.org/10.1145/2635868.2635929

[30] James Kennedy and Russell Eberhart. 1995. Particle swarm optimization. In *Proceedings of International Conference on Neural Networks (ICNN '95)*. IEEE, 1942–1948. DOI: https://doi.org/10.1109/ICNN.1995.488968

[31] Nabeel Ali Khan, Sadiq Ali, and Kwonhue Choi. 2021. An instantaneous frequency and group delay based feature for classifying EEG signals. *Biomed. Signal Process. Control* 67 (2021), 102562.

[32] Meghdad Khazaee, Ahmad Banakar, Barat Ghobadian, Mostafa Agha Mirsalim, and Saeid Minaei. 2021. Remaining useful life (RUL) prediction of internal combustion engine timing belt based on vibration signals and artificial neural network. *Neural Comput. Appl.* 33, 13 (2021), 7785–7801. DOI: https://doi.org/10.1007/s00521-020-05520-3

[33] Deok-Jae Kwon, Jun-Hyuk Im, Mudassir Raza Siddiqi, and Jin Hur. 2020. Detection technique for manufacturing imperfection of rare-earth magnets on IPMSM. In *Proceedings of the 2020 IEEE Energy Conversion Congress and Exposition (ECCE '20)*. IEEE, 1407–1410. DOI: https://doi.org/10.1109/ECCE44975.2020.9235734

[34] Bing Liu, Shiva Nejati, Lucia, and Lionel C. Briand. 2019. Effective fault localization of automotive simulink models: Achieving the trade-off between test Oracle effort and fault localization accuracy. *Empir. Softw. Eng.* 24, 1 (2019), 444–490. DOI: https://doi.org/10.1007/s10664-018-9611-z

[35] Oded Maler and Dejan Nickovic. 2004. Monitoring temporal properties of continuous signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems, Joint International Conferences on Formal Modelling and Analysis of Timed Systems (FORMATS '04) and Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT '04)*. Yassine Lakhnech and Sergio Yovine (Eds.), Lecture Notes in Computer Science, Vol. 3253, Springer, Berlin, 152–166. DOI: https://doi.org/10.1007/978-3-540-30206-3_12

[36] Chengying Mao, Xuzheng Zhan, Jinfu Chen, Jifu Chen, and Rubing Huang. 2020. Adaptive random testing based on flexible partitioning. *IET Softw.* 14, 5 (2020), 493–505. DOI: https://doi.org/10.1049/IET-SEN.2019.0325

[37] Logan Mathesen, Shakiba Yaghoubi, Giulia Pedrielli, and Georgios Fainekos. 2019. Falsification of cyber-physical systems with robustness uncertainty quantification through stochastic optimization with adaptive restart. In *Proceedings of the 15th IEEE International Conference on Automation Science and Engineering (CASE '19)*. IEEE, 991–997. DOI: https://doi.org/10.1109/COASE.2019.8843005

[38] Mathworks. 2023. Detect faults in aircraft elevator control system. Mathworks. Retrieved July 20, 2023 from https://in.mathworks.com/help/stateflow/ug/fault-detection-control-logic-in-an-aircraft-elevator-control-system.html

[39] Mathworks. 2023. Model a fault-tolerant fuel control system. Mathworks. Retrieved July 20, 2023 from https://in.mathworks.com/help/simulink/slref/modeling-a-fault-tolerant-fuel-control-system.html

[40] Mathworks. 2023. Model an automatic transmission controller. Mathworks. Retrieved July 20, 2023 from https://in.mathworks.com/help/simulink/slref/modeling-an-automatic-transmission-controller.html

[41] Mathworks. 2023. Signal features. Mathworks. Retrieved July 20, 2023 from https://in.mathworks.com/help/predmaint/ug/signal-features.html

[42] Mathworks. 2023. Simulink documentation. Mathworks. Retrieved July 20, 2023 from https://in.mathworks.com/help/simulink/

[43] Mathworks. 2023. Structural coverage metrics. Mathworks. Retrieved July 20, 2023 from https://in.mathworks.com/help/slcoverage/gs/coverage-terminology.html

[44] Mathworks. 2024. Design NARMA-L2 neural controller in simulink. Mathworks. Retrieved May 13, 2024 from https://au.mathworks.com/help/deeplearning/ug/design-narma-l2-neural-controller-in-simulink.html

[45] Reza Matinnejad, Shiva Nejati, Lionel C. Briand, and Thomas Bruckmann. 2015. Effective test suites for mixed discrete-continuous stateflow controllers. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE '15)*. Elisabetta Di Nitto, Mark Harman, and Patrick Heymans (Eds.), ACM, 84–95. DOI: https://doi.org/10.1145/2786805.2786818

[46] Reza Matinnejad, Shiva Nejati, Lionel C. Briand, and Thomas Bruckmann. 2019. Test generation and test prioritization for simulink models with dynamic behavior. *IEEE Trans. Softw. Eng.* 45, 9 (2019), 919–944. DOI: https://doi.org/10.1109/TSE.2018.2811489

[47] Anastasia Mavridou, Hamza Bourbouh, Dimitra Giannakopoulou, Thomas Pressburger, Mohammad Hejase, Pierre-Loïc Garoche, and Johann Schumann. 2020. The ten lockheed Martin cyber-physical challenges: Formalized, analyzed, and explained. In *Proceedings of the 28th IEEE International Requirements Engineering Conference (RE '20)*. Travis D. Breaux, Andrea Zisman, Samuel Fricker, and Martin Glinz (Eds.). IEEE, 300–310. DOI: https://doi.org/10.1109/RE48521.2020.00040

[48] Claudio Menghi, Paolo Arcaini, Walstan Baptista, Gidon Ernst, Georgios Fainekos, Federico Formica, Sauvik Gon, Tanmay Khandait, Atanu Kundu, Giulia Pedrielli, et al. 2023. ARCH-COMP23 category report: Falsification. In *Proceedings of 10th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH '23)*. Goran Frehse and Matthias Althoff (Eds.), EPiC Series in Computing, Vol. 96, EasyChair, 151–169. DOI: https://doi.org/10.29007/6nqs

[49] Anitha Murugesan, Michael W. Whalen, Neha Rungta, Oksana Tkachuk, Suzette Person, Mats Per Erik Heimdahl, and Dongjiang You. 2015. Are we there yet? Determining the adequacy of formalized requirements and test suites. In *Proceedings of the7th International Symposium on NASA Formal Methods (NFM '15)*. Klaus Havelund, Gerard J. Holzmann, and Rajeev Joshi (Eds.), Lecture Notes in Computer Science, Vol. 9058, Springer, Cham, 279–294. DOI: https://doi.org/10.1007/978-3-319-17524-9_20

[50] Akbar Siami Namin and James H. Andrews. 2009. The influence of size and coverage on test suite effectiveness. In *Proceedings of the 18th International Symposium on Software Testing and Analysis (ISSTA '09)*. Gregg Rothermel and Laura K. Dillon (Eds.), ACM, New York, NY, 57–68. DOI: https://doi.org/10.1145/1572272.1572280

[51] Shiva Nejati, Khouloud Gaaloul, Claudio Menghi, Lionel C. Briand, Stephen Foster, and David Wolfe. 2019. Evaluating model testing and model checking for finding requirements violations in simulink models. In *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/SIGSOFT FSE '19)*. Marlon Dumas, Dietmar Pfahl, Sven Apel, and Alessandra Russo (Eds.), ACM, 1015–1025. DOI: https://doi.org/10.1145/3338906.3340444

[52] Luan Viet Nguyen, James Kapinski, Xiaoqing Jin, Jyotirmoy V. Deshmukh, and Taylor T. Johnson. 2017. Hyperproperties of real-valued signals. In *Proceedings of the 15th ACM-IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE '17)*. Jean-Pierre Talpin, Patricia Derler, and Klaus Schneider (Eds.), ACM, 104–113. DOI: https://doi.org/10.1145/3127041.3127058

[53] Dejan Nickovic and Tomoya Yamaguchi. 2020. RTAMT: Online robustness monitors from STL. In *Proceedings of the 18th International Symposium on Automated Technology for Verification and Analysis (ATVA '20)*. Dang Van Hung and Oleg Sokolsky (Eds.), Lecture Notes in Computer Science, Vol. 12302, Springer, Berlin, 564–571. DOI: https://doi.org/10.1007/978-3-030-59152-6_34

[54] Jungsup Oh, Mark Harman, and Shin Yoo. 2011. Transition coverage testing for simulink/stateflow models using messy genetic algorithms. In *Proceedings of the 13th Annual Genetic and Evolutionary Computation Conference (GECCO '11)*. Natalio Krasnogor and Pier Luca Lanzi (Eds.), ACM, 1851–1858. DOI: https://doi.org/10.1145/2001576.2001825

[55] Giulia Pedrielli, Tanmay Khandait, Yumeng Cao, Quinn Thibeault, Hao Huang, Mauricio Castillo-Effen, and Georgios Fainekos. 2024. Part-X: A family of stochastic algorithms for search-based test generation with probabilistic guarantees. *IEEE Trans Autom. Sci. Eng.* 21, 3 (2024), 4504–4525. DOI: https://doi.org/10.1109/TASE.2023.3297984

[56] Kalaivani Rathakrishnan, Seung-Nam Min, and Se Jin Park. 2020. Evaluation of ECG features for the classification of post-stroke survivors with a diagnostic approach. *Appl. Sci.* 11, 1 (2020), 192.

[57] Zion Market Research. 2023. Cyber-physical systems (CPS) market size, share, trends & research 2030. Zion Market Research. Retrieved July 20, 2023 from https://www.zionmarketresearch.com/report/cyber-physical-systems-market

[58] Masume Saljuqi and Peyvand Ghaderyan. 2023. Combining homomorphic filtering and recurrent neural network in gait signal analysis for neurodegenerative diseases detection. *Biocybern. Biomed. Eng.* 43, 2 (2023), 476–493.

[59] Sohil Lal Shrestha. 2020. Automatic generation of simulink models to find bugs in a cyber-physical system tool chain using deep learning. In *Proceedings of the 42nd International Conference on Software Engineering (ICSE '20)*. Gregg Rothermel and Doo-Hwan Bae (Eds.), Companion Volume, ACM, 110–112. DOI: https://doi.org/10.1145/3377812.3382163

[60] Sohil Lal Shrestha, Shafiul Azam Chowdhury, and Christoph Csallner. 2022. SLNET: A redistributable corpus of 3rd-party simulink models. In *Proceedings of the 19th IEEE/ACM International Conference on Mining Software Repositories (MSR '22)*. ACM, 1–5. DOI: https://doi.org/10.1145/3524842.3528001

[61] Nikhil Kumar Singh and Indranil Saha. 2020. Specification-guided automated debugging of CPS models. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 39, 11 (2020), 4142–4153. DOI: https://doi.org/10.1109/TCAD.2020.3012862

[62] Lev Sorokin, Tiziano Munaro, Damir Safin, Brian Hsuan-Cheng Liao, and Adam Molin. 2023. OpenSBT: A modular framework for search-based testing of automated driving systems. arXiv:2306.10296. DOI: https://doi.org/10.48550/arXiv.2306.10296

[63] Matt Staats and Corina S. Pasareanu. 2010. Parallel symbolic execution for structural test generation. In *Proceedings of the 19th International Symposium on Software Testing and Analysis (ISSTA '10)*. Paolo Tonella and Alessandro Orso (Eds.). ACM, 183–194. DOI: https://doi.org/10.1145/1831708.1831732

[64] Toru Takisaka, Zhenya Zhang, Paolo Arcaini, and Ichiro Hasuo. 2023. A robustness-based confidence measure for hybrid system falsification. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 42, 5 (2023), 1718–1731. DOI: https://doi.org/10.1109/TCAD.2022.3201157

[65] Manel Tekaya, Mohamed Taha Bennani, Mohamed Abidi Alagui, and Samir Ben Ahmed. 2015. Aspect-oriented test case generation from matlab/simulink models. In *Proceedings of the 10th International Conference on Dependability and Complex Systems on Theory and Engineering of Complex Systems and Dependability* (DepCoS-RELCOMEX). Wojciech Zamojski, Jacek Mazurkiewicz, Jaroslaw Sugier, Tomasz Walkowiak, and Janusz Kacprzyk (Eds.), Advances in Intelligent Systems and Computing, Vol. 365, Springer, Cham, 495–504. DOI: https://doi.org/10.1007/978-3-319-19216-1_47

[66] Quinn Thibeault, Jacob Anderson, Aniruddh Chandratre, Giulia Pedrielli, and Georgios Fainekos. 2021. PSY-TaLiRo: A Python toolbox for search-based test generation for cyber-physical systems. In *Proceedings of the 26th International Conference on Formal Methods for Industrial Critical Systems (FMICS '21)*. Alberto Lluch-Lafuente and Anastasia Mavridou (Eds.), Lecture Notes in Computer Science, Vol. 12863, Springer, Cham, 223–231. DOI: https://doi.org/10.1007/978-3-030-85248-1_15

[67] Shakiba Yaghoubi and Georgios Fainekos. 2019. Gray-box adversarial testing for control systems with machine learning components. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control (HSCC '19)*. Necmiye Ozay and Pavithra Prabhakar (Eds.), ACM, USA, 179–184. DOI: https://doi.org/10.1145/3302504.3311814

[68] Orhan Yaman, Fatih Ertam, and Turker Tuncer. 2020. Automated Parkinson's disease recognition based on statistical pooling method using acoustic features. *Med. Hypotheses* 135 (2020), 109483.

[69] Qian Yang, J. Jenny Li, and David M. Weiss. 2006. A survey of coverage based testing tools. In *Proceedings of the 2006 International Workshop on Automation of Software Test (AST '06)*. Hong Zhu, Joseph R. Horgan, Shing-Chi Cheung, and J. Jenny Li (Eds.), ACM, 99–103. DOI: https://doi.org/10.1145/1138929.1138949

[70] Justyna Zander-Nowicka. 2009. *Model-Based Testing of Real-Time Embedded Systems in the Automotive Domain*. Technical University Berlin, Berlin, Germany.

[71] Yuan Zhan and John A. Clark. 2008. A search-based framework for automatic testing of MATLAB/simulink models. *J. Syst. Softw.* 81, 2 (2008), 262–285. DOI: https://doi.org/10.1016/j.jss.2007.05.039

Appendices

A Results Considering Variance for Signal Feature Coverage

A.1 Comprehensive Signal Feature Coverage

Figure A1 shows the trends over time of ζ_{comp} for various test generation techniques. We present a tubular visualization showcasing the ζ_{comp}, incorporating minimum, maximum, and mean values. The boundaries of the tubes are defined by the minimum and maximum values, with the mean value depicted by a thick line encapsulated within these boundaries.

A.2 GT Signal Feature Coverage

Figure A2 shows the trends of ζ_{ϕ} (computed for GT features and signals) obtained by different test generation techniques. Similar to Figure A1, this tubular visualization shows minimum, maximum, and mean values of ζ_{ϕ}. We note that all the tubes are tightly packed, indicating that there is not a significant spread in the data. For more detailed information on the data, we encourage the readers to visit our online repository.

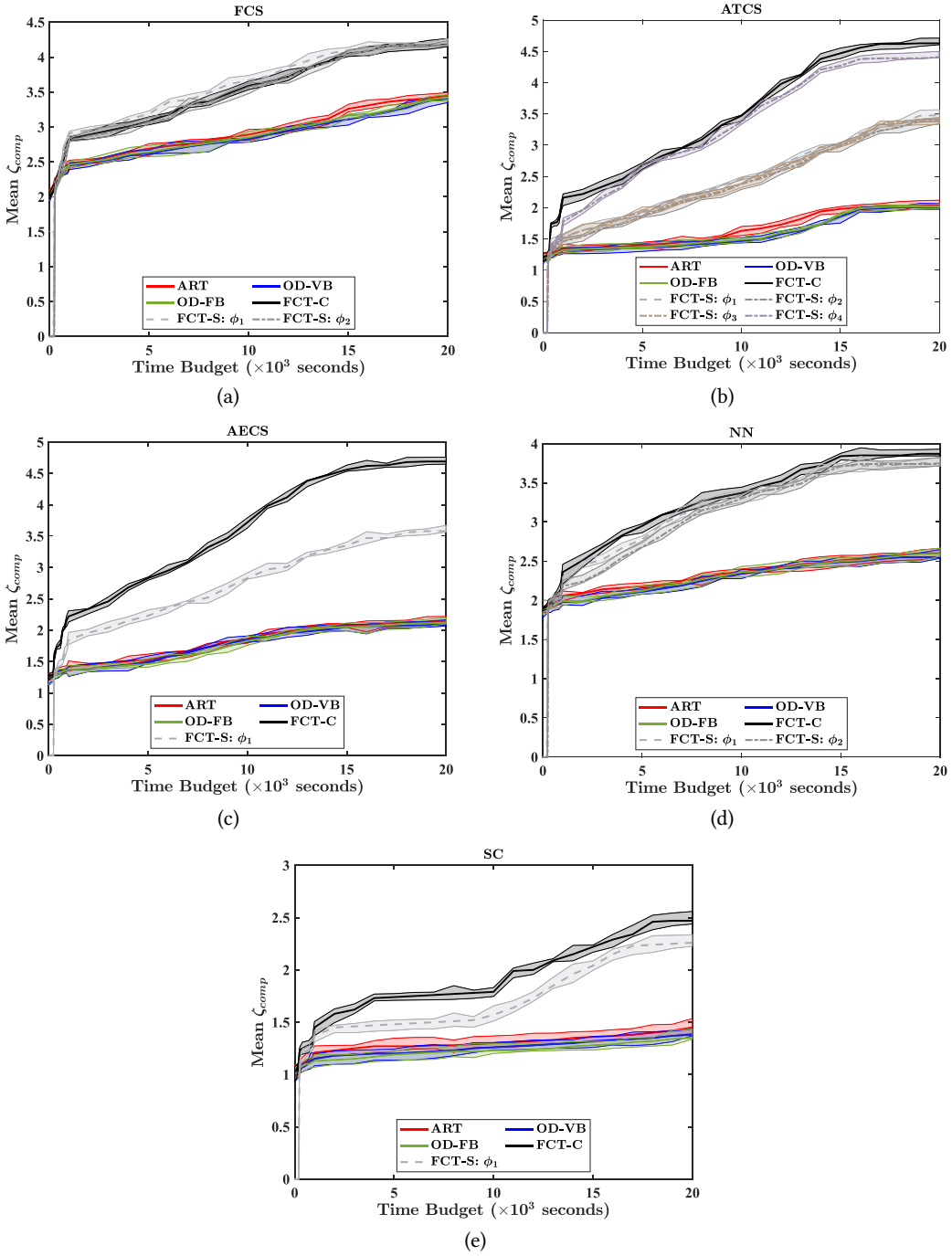


Fig. A1. Trends over time of ζ_{comp} achieved by different test generation strategies.

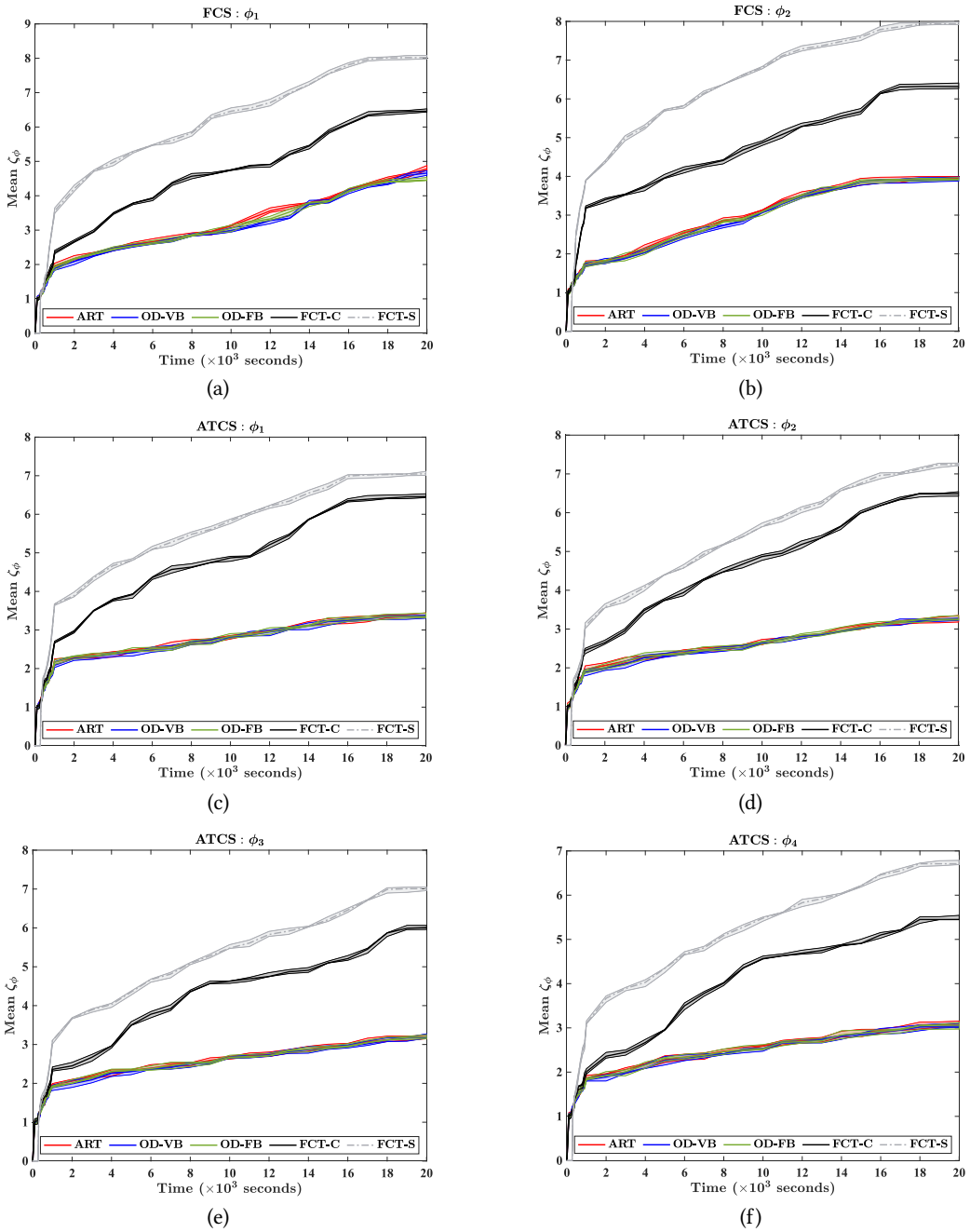


Fig. A2. Trends over time of ζ_ϕ achieved by different test generation strategies.

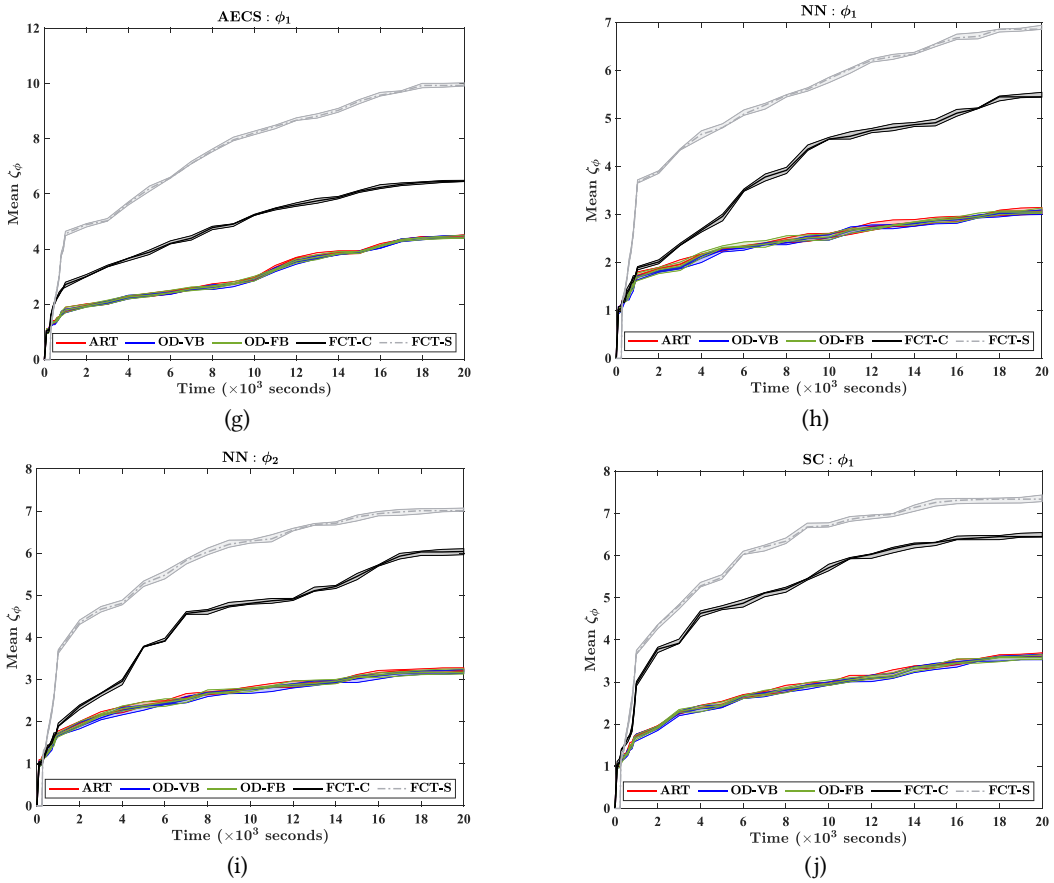


Fig. A2. Continued

B Results Considering Variance for PBMT

In Figure B1, we show the trends over time of the number of mutants ϕ-killed w.r.t. a given property ϕ, incorporating minimum, maximum, and mean values. This visualization offers a comprehensive representation of the variability in mutant killing across different scenarios, allowing for a clearer understanding of the overall performance.

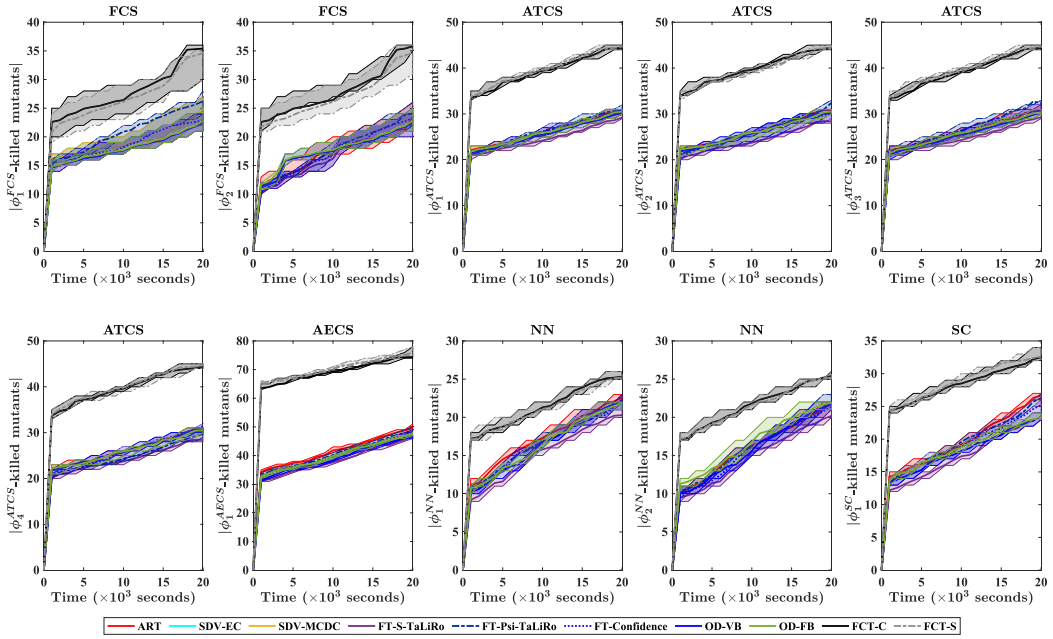


Fig. B1. Trends over time of the number of ϕ-killed mutants by different test generation strategies.

Received 14 May 2024; revised 27 November 2024; accepted 6 January 2025