Fault Injection for Simulink-based CPS Models: Insights and Future Directions

Drishti Yadav
University of Luxembourg
Luxembourg
drishti.yadav@uni.lu

Claudio Mandrioli
University of Luxembourg
Luxembourg
claudio.mandrioli@uni.lu

Ezio Bartocci
TU Wien
Vienna, Austria
ezio.bartocci@tuwien.ac.at

Domenico Bianculli
University of Luxembourg
Luxembourg
domenico.bianculli@uni.lu

***Abstract*—Ensuring the safety and reliability of Cyber-Physical Systems (CPS) is critical, particularly in safety-critical domains such as automotive and aerospace. Fault Injection (FI) is a well-established technique for testing system resilience, but current FI tools often face challenges when applied to Simulink-based CPS models. In this paper, we analyze the shortcomings of existing FI methods, and reflect on the key challenges of FI for Simulink-based CPS models. By offering insights into these challenges and proposing research pathways, we aim to inspire further advances in FI methodologies, enabling more robust testing of CPS in real-world applications.**

***Index Terms*—Cyber-Physical Systems, Fault Injection, Simulink, Model-Based Development, Testing, Mutation**

I. Introduction

Cyber-Physical Systems (CPS) have become integral to a wide range of critical applications, including autonomous vehicles, industrial automation, and healthcare systems. These systems combine both physical components and computational elements, creating complex interactions that require rigorous verification and validation (V&V) to ensure reliability and safety. One key aspect of this V&V process is fault injection (FI), a technique used to simulate system failures and assess how well a CPS can handle and recover from faults [1].

In the context of CPS, Model-Based System Development using *Simulink* (registered trademark of The MathWorks, Inc.) has become the de facto standard for designing and testing them [2, 3]. Simulink provides a powerful simulation environment that allows engineers to model, simulate, and analyze the behavior of the system before deployment. Despite significant advancements in FI methodologies for traditional software systems, their application to Simulink-based CPS models has not yet been fully explored. Unlike conventional software, Simulink models exhibit continuous-time behaviors, dynamic interactions, and hybrid system properties, making traditional approaches to FI unsuitable.

In this paper, we reflect on the current state of FI for Simulink-based CPS models, highlighting key gaps and challenges in existing methodologies. Drawing from these reflections, we propose research directions that can help overcome these challenges. We reflect on these challenges and envisioning potential solutions, this *reflection paper* seeks to contribute to the ongoing discourse on improving the FI process for CPS, thereby fostering more robust system designs and supporting broader automation in software engineering practices.

The paper is organized as follows. Section II provides the background on FI and state-of-the-art tools for Simulink models. Section III presents our critical review of the existing FI techniques and tools, highlighting their main features and limitations. Section IV discusses key challenges and proposes future research directions. Section V concludes the paper.

II. Background & Related Work

FI is a well-established method for evaluating system resilience by introducing faults and observing system behavior in presence of failures. Conceptually, it is similar to *Mutation Testing* (MT) [4], where syntactic changes (mutations) are introduced into the software to assess the quality of a test suite. In MT, faulty programs (aka *mutants*) are typically created by altering the original program's code (e.g., changing operators or variables), and the test suite is evaluated based on its ability to detect these alterations [4, 5]. Previous research used FI on CPS Simulink models to aid V&V [6, 7].

A. Key requirements for effective FI

Researchers have identified several essential criteria for FI in Simulink-based CPS models to be effective [1, 8]. Among them, we highlight the following:

(i) *Automation:* The process should run without manual intervention, enabling efficient and effective testing.
(ii) *Comprehensive Fault Coverage:* A diverse set of faults and fault locations should be supported to evaluate system behavior under various failure conditions.
(iii) *Seamless Compatibility:* It should integrate smoothly with existing simulation environments and testing frameworks to improve usability.
(iv) *Consistent and Repeatable Results:* FI experiments must produce reliable and reproducible results to ensure meaningful analysis.

B. State-of-the-art FI tools

Many FI tools have been developed for Simulink models [9, 10, 11, 12], each offering different features and approaches. However, prior studies have highlighted drawbacks and constraints associated with some of these tools [1, 3].

- MODIFI [10] and ErrorSim [11] provide mechanisms for injecting faults into Simulink-based systems, although they support a limited range of fault types and are not available.

2643-1572/25/$31.00 ©2025 IEEE
DOI 10.1109/ASE63991.2025.00326

- SIMULTATE [9] integrates Python and MATLAB to facilitate FI and MT through an interactive user interface. While this approach is beneficial for users unfamiliar with Simulink, manually configuring FI scenarios may pose challenges when scaling experiments to inject a large number of faults.
- Fabarisov et al. [12] introduce a model-based FI technique that enables the injection of various fault types into Simulink models. However, it does not provide automated placement of fault blocks within the system under test. This *(i)* requires additional manual effort for configuration and *(ii)* limits its scalability and usability.
- Moradi et al. [13] present an automated FI framework for both model-in-the-loop and hardware-in-the-loop (HiL) simulations. One of the drawbacks of this framework is that it allows injecting only one fault per model at a time.
- Ramírez Bárcenas et al. [14] introduce another model-based FI framework that allows injecting faults at different points in the system. However, the FI process is not automated and the tool is not publicly available.
- Simulink Fault Analyzer [15], developed by MathWorks, provides a graphical interface for defining fault scenarios and integrates seamlessly with MathWorks' Simulink Test and Requirements Toolbox. However, its configuration is manual, and with limited predefined fault types—although it allows the definition of custom fault types.
- FIM [1] provides a more automated approach to FI, supporting single and multifault injection with a wide variety of faults and mutation operators. However, it lacks *(i)* support for injecting faults in Stateflow charts and *(ii)* features for post-FI activities such as fault propagation analysis.
- BERTiMuS [16] is a tool that leverages a large language model (LLM) to mutate Simulink models. It transforms graphical Simulink representations into a textual format, allowing the LLM to analyze and modify the model efficiently. Although it supports FI at the block level and faults in Stateflow charts, it lacks fault injection in the signals.
- MUT4SLX [17, 18] is a MT tool designed for Simulink, offering a diverse set of mutation operators. Similar to BERTiMuS, it excels in block-level FI and supports faults in Stateflow charts, but lacks support for signal-level FI.

III. Critical Review of State-of-the-art FI Tools

We compare the state-of-the-art FI tools for Simulink-based CPS models in Table I, where the evaluation criteria are organized into two broad aspects: **Features** and **Faults Support**. The **Features** aspect is further broken down into the following four dimensions:

1) Availability, as a basic requirement to enable further research. Column *Available* indicates whether the tool is (**D**) publicly available for download (but without license), (**OS**) open-source (downloadable with license), or (**C**) commercially available.
2) Ability to automate the FI process. Column *Automated* indicates whether the tool is able to automatically insert faults in the model, reducing manual intervention.
3) Ability to inject multiple faults and manage their triggering with a rich timing model, indicated through two columns:
   - *Multi-FI* indicates if the tool supports multiple fault injections simultaneously, which is needed for testing complex[1] CPS with multiple potential failure points.
   - *Fault-Timing* indicates whether the tool timing of the injected faults is (**F**) fixed (the fault is always present), (**D**) deterministic (the fault happens at a fixed time), (**P**) probabilistic (faults are triggered at a random time).
4) Features that can enable higher levels of automation of the FI process, indicated through the following four columns:
   - *Coverage-based* indicates if the tool supports coverage-based FI, providing the user with some model coverage metric achieved by the injected faults.
   - *Syntactic Aware* indicates whether the tool is aware of the structure or syntax of the system being tested (e.g., if it can understand the types of signals to prevent the creation of broken models).
   - *Semantic Aware* indicates whether the tool understands the semantics or behavior of the system, meaning it can intelligently place faults based on the functional context and interactions of the system components.
   - *Analysis Features* indicates whether the tool provides some form of (**P**) fault propagation tracking, minimal (**C**) cut-set analysis, or automated (**R**) reporting, which help engineers understand how faults affect the CPS.

For each tool, the **Features** columns in Table I report whether it satisfies the criteria (✓) or not (×). For the *Available*, *Fault-Timing* and *Analysis Features* columns, we use the categories described above, D/OS/C, F/D/P, and P/C/R, respectively.

To evaluate the support for different fault types by existing FI tools, the **Faults Support** columns in Table I categorize the tools based on the following criteria:

- *Signal-based* indicates whether the tool supports FI in *lines* or *connections* within the model, simulating faults in data flow between blocks by modifying the signal, e.g., with "Absolute" or "Negate" blocks, or by adding noise.
- *Block-based* indicates whether the tool supports block mutations, i.e., FI at the block-level by modifying the block's functionality, such as modifying arithmetic or logical operators, or other block-level components.
- *Stateflow* indicates whether the tool supports FI within Stateflow charts, e.g., incorrect actions or transitions.
- *Look-Up Table (LUT)*[2] *support* indicates whether the tool supports FI in LUTs.

Based on Table I, we observe the following limitations in the state-of-the-art FI tools:

[1]Here, with complexity, we refer to a combination of structural and semantic aspects, not just the number of blocks or lines. Hierarchical depth (i.e., nesting levels) and control logic, including state machines/Stateflow charts, signal routing, feedback loops, control flow constructs (such as enabled, triggered, or state-dependent subsystems), all contribute to complexity.

[2]LUTs are commonly used to store hard-coded parameters of industrial hybrid systems. Any errors in these tables can lead to significant system failures or performance degradation.

TABLE I
COMPARISON OF STATE-OF-THE-ART FI TOOLS FOR SIMULINK-BASED CPS MODELS

Tools	Features								Faults Support			
	Available	Automated	Multi-FI	Fault-Timing	Coverage-based	Syntactic Aware	Semantic Aware	Analysis Features	Signal-based	Block-based	Stateflow	LUT support
MODIFI [10]	×	×	✓	P	×	✓	×	C	✓	×	×	×
ErrorSim [11]	×	×	✓	P	×	×	×	P	✓	×	×	×
SIMULTATE [9]	D	×	✓	DP	×	×	×	×	✓	✓	×	×
FIBlock [12]	OS	×	✓	P	×	×	×	×	✓	×	×	×
[13]	×	×	×	D	×	×	×	×	✓	×	×	×
[14]	×	×	✓	F	×	×	×	×	✓	×	×	×
Simulink Fault Analyzer [15]	C	×	✓	DP	×	×	×	×	✓	×	×	×
FIM [1]	OS	✓	✓	DP	×	×	×	×	✓	✓	×	×
BERTiMuS [16]	D	✓	×	F	×	×	✓	×	×	✓	✓	×
MUT4SLX [17, 18]	OS	✓	×	F	×	✓	×	R	×	✓	✓	×

L1 Most tools offer little automation, lacking support for system-wide multi-fault injection and often requiring extensive custom scripting.

L2 Some tools support deterministic faults, while other tools also offer probabilistic faults. However, not all tools provide rich timing options, such as stochastic fault events or Mean Time to Failure (MTTF) modeling, and no tool supports automated fault timing selection, limiting the ability to efficiently explore diverse fault timings.

L3 No tool supports fault selection based on Simulink model coverage criteria.

L4 The majority of the tools lack syntactic model awareness, ignoring, for example, block compatibility and type requirements, leading to structural invalidity. As a consequence, FI can result in *broken* or *invalid* mutants that fail to compile (for instance, when replacing relational with logical operator), and waste computational resources.

L5 Almost all tools lack semantic awareness. The only exception is BERTiMuS [16], which uses a textual format representation of the models to inject the faults. Without semantic awareness, injecting some type of faults into specific parts of the model can lead to the creation of "equivalent mutants" (borrowing the terminology from MT) that do not affect the behavior of the system, making the fault injection ineffective. For example, in the work of Bartocci et al. [7], "Absolute" faults were injected in the Simulink model of an Automatic Transmission Controller System into some connections that inherently do not propagate negative values. As a result, the faults had no effect, as the positive values remained positive.

L6 Existing tools offer limited fault analysis, specifically, fault propagation tracking and analysis, making it challenging to analyze cascading failures.

L7 Only two tools support both signal-based and block-based FI. Injecting faults of these classes requires different approaches: intuitively, adding noise to a signal (signal-based FI) is different from swapping two blocks (block-based FI). However, supporting both fault types is important to improve coverage of fault types.

L8 Only two tools support FI for Stateflow charts, and no tool supports LUT fault injection. In prior work [7], the authors have explored two fault operations for LUTs: "Stuck-at 0" for a single entry, and "swapping two neighboring entries". However, these fault operations are not directly integrated into existing tools. This highlights that current FI tools do not adequately address the potential vulnerabilities in Stateflow charts and LUTs.

IV. CHALLENGES IN CPS FI AND THE PATH FORWARD

Based on the observations made in the previous section, we now reflect on the critical challenges in FI for CPS Simulink models and propose future research directions to address them.

(1) Automation and Configuration complexity. Many FI tools require significant manual configuration, forcing engineers to rely on custom scripting to set up FI configurations (such as multi-fault injection), increasing time and effort (**L1**). Additionally, lack of automation makes FI time-intensive for engineers, as they must manually identify high-risk components and configure faults without built-in support for systematic fault selection. Without automation, FI in complex CPS remains a resource-intensive and error-prone process.

Future directions: To enhance FI automation, we recommend conducting research on (i) selection of signals and blocks where to inject faults, (ii) generation of the faults (either template-based or template-free), and (iii) fault injection and setup, while minimizing manual configuration. For example, agentic AI-driven fault selection algorithms could automatically generate FI configurations based on model behavior, which would be especially beneficial for large, heterogeneous models that are difficult to configure manually.

(2) Lack of rich fault timing adjustments. Not all FI tools support diverse fault timings (**L2**), limiting their flexibility to (i) model realistic and diverse fault behaviors, and (ii) test unpredictable faults that occur randomly in practical applications. Moreover, there is no support to identify the optimal fault activation time (τ), i.e., the time at which triggering a fault would lead to observable changes in the system output or result in system failures. Currently, all FI tools either rely on

randomly selecting fault activation times or employ exhaustive FI by activating faults at all possible timestamps, which is both inefficient and computationally expensive.
Future directions: FI tools should support probabilistic fault timings to simulate real-world failure behaviors. They should also provide support for automated search techniques to identify the most effective τ. This would enable more efficient and targeted FI, eliminating the need to randomly choose τ and reducing the cost of performing exhaustive FI.

(3) Coverage-based injection limitations. Large and complex CPS models, particularly in the automotive and aerospace industries, contain hundreds of interconnected blocks (e.g., >600 blocks in three Simulink models developed by Delphi Automotive Systems [19], and >800 blocks in an aircraft model developed by MathWorks [7]) making exhaustive FI infeasible. Currently, all existing FI tools lack a systematic coverage-driven approach to FI (**L3**). Instead, they rely on random FI or predefined fault libraries, which may fail to systematically target critical system behaviors. Current tools do not align FI with coverage goals, e.g., with Simulink model coverage criteria, such as execution or decision coverage, leading to gaps in fault testing and reducing confidence in resilience assessments. While coverage-based FI exist for traditional software [20], migrating it to Simulink CPS models is non-trivial due to hybrid dynamics and model-specific semantics, as shown in prior work proposing *signal feature coverage* for CPS Simulink models [21].
Future directions: Future research should introduce model-driven coverage criteria for FI, which will ensure that testing is thorough and comprehensive. This approach will enable FI tools to systematically identify and target uncovered, insufficiently tested or high-risk areas of the system, ensuring that FI is aligned with the most critical behaviors of the system.

(4) Lack of model awareness. Existing FI tools often treat Simulink models as black boxes, injecting faults without understanding their syntax and semantics (**L4–5**). Without syntactic and semantic awareness, FI tools may inject faults into irrelevant components, resulting in either *(i)* broken or invalid models that fail to compile, or *(ii)* faults with no observable impact. This limitation leads to a waste of resources, missed vulnerabilities, and reduces the overall effectiveness of V&V.
Future directions: Future FI tools should leverage the syntax and semantics of the Simulink model to enable intelligent, context-sensitive FI. This involves understanding the underlying structure (syntactic awareness) and behavior (semantic awareness) of the model. In this way, faults can be injected into system components that are critical for CPS reliability, ensuring comprehensive testing while avoiding the injection of faults into irrelevant or non-critical components.

(5) Fault Analysis. Another challenge is the limited support of existing tools for tracking and visualizing fault propagation across interconnected subsystems (**L6**). Without effective fault propagation tracking, it is difficult to trace how an injected fault spreads through components, making debugging and failure analysis expensive, especially in complex models, where multiple faults may interact in non-trivial ways. Current tools that include some analysis features are limited to analyzing individual faults in isolation, and do not consider multiple simultaneous faults. This limits the assessment of combined effects due to interactions between multiple active faults.
Future directions: Future research should focus on developing visualization techniques to trace and analyze fault propagation across subsystems. This includes AI-driven or graph-based fault propagation analysis [22, 23] that helps (i) track how injected faults affect interconnected components, and (ii) map fault paths to predict which component failures could cause a ripple effect, leading to system-wide issues. Further, by *enabling* and *automating* the injection and analysis of single and multiple faults, tools can provide deeper insights into the system's robustness and its ability to handle simultaneous failures, improving the testing efficiency and the overall evaluation of fault resilience. This will also help prioritize testing by identifying which fault combinations are most likely to lead to system failures and require further investigation.

(6) Limited fault models for Simulink components. Many existing FI tools focus on either blocks or signal flow faults, failing to address both of them simultaneously (**L7**). Furthermore, none of them supports FI in complex LUTs, such as n-D LUTs and Prelookup blocks. These blocks play an essential role in model-based CPS development by modeling critical system behaviors, such as friction coefficients (as in [24]) and aerodynamic coefficients (as in [25]). Finally, many tools fail to address time-dependent elements (such as Stateflow charts, discrete-time systems, switching components), limiting their applicability to industrial use cases (**L8**).
Future directions: We recommend extending FI support to critical Simulink components, covering both signal- and block-based faults, and LUTs and state machines. Furthermore, research should focus on improving fault models to reflect real-world episodes (including environmental variations and uncertainties), ensuring that simulation results are more aligned with actual CPS operation environments.

V. Conclusions

In this paper, we have explored the current state of FI for CPS Simulink models, and highlighted the need for next-generation FI solutions with automated mechanisms for fault selection and injection, coverage-based injection capabilities, syntactic and semantic awareness, analysis features, and richer fault models. Future research should focus on smarter, automated, scalable, coverage-driven and model-aware FI solutions to enhance their effectiveness in safety-critical domains.

This paper may serve as a starting point for a dialogue between industry professionals and researchers working on CPS V&V [26], motivating research to overcome existing challenges and bring next-generation FI solutions to fruition. Addressing these issues is crucial for advancing CPS reliability and robustness in increasingly complex applications.

Acknowledgments

This project has received funding from the European Union's Horizon Europe program under Grant Agreement No. 101148870 (ContTestCPS).

References

[1] E. Bartocci, L. Mariani, D. Nickovic, and D. Yadav, "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 2022, Singapore, Singapore, November 14-18, 2022*, A. Roychoudhury, C. Cadar, and M. Kim, Eds. USA: ACM, 2022, pp. 1716–1720. [Online]. Available: https://doi.org/10.1145/3540250.3558932

[2] Mathworks. (2025) Simulink documentation. Mathworks. [Online]. Available: https://in.mathworks.com/help/simulink/

[3] D. Yadav, "From fault injection to formal verification: A holistic approach to fault diagnosis in cyber-physical systems," in *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2024. New York, NY, USA: ACM, 2024, p. 1896–1900. [Online]. Available: https://doi.org/10.1145/3650212.3685552

[4] R. A. DeMillo, R. J. Lipton, and F. G. Sayward, "Hints on test data selection: Help for the practicing programmer," *Computer*, vol. 11, no. 4, pp. 34–41, April 1978.

[5] A. T. Acree, T. A. Budd, R. A. DeMillo, R. J. Lipton, and F. G. Sayward, "Mutation analysis," Georgia Institute of Technology, Atlanta, Georgia, techreport GIT-ICS-79/08, 1979.

[6] E. Bartocci, L. Mariani, D. Nickovic, and D. Yadav, "Search-based testing for accurate fault localization in CPS," in *IEEE 33rd International Symposium on Software Reliability Engineering, ISSRE 2022, Charlotte, NC, USA, October 31 - Nov. 3, 2022*. USA: IEEE, 2022, pp. 145–156. [Online]. Available: https://doi.org/10.1109/ISSRE55969.2022.00024

[7] E. Bartocci, L. Mariani, D. Nickovic, and D. Yadav, "Property-based mutation testing," in *IEEE Conference on Software Testing, Verification and Validation, ICST 2023, Dublin, Ireland, April 16-20, 2023*. USA: IEEE, 2023, pp. 222–233. [Online]. Available: https://doi.org/10.1109/ICST57152.2023.00029

[8] D. Yadav, "Enhancing fault diagnosis in safety-critical cyber-physical systems," Ph.D. dissertation, Technische Universität Wien, Vienna, Austria, 2025. [Online]. Available: https://doi.org/10.34726/hss.2025.125790

[9] I. Pill, I. Rubil, F. Wotawa, and M. Nica, "Simultate: A toolset for fault injection and mutation testing of simulink models," in *2016 IEEE Ninth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. USA: IEEE, 2016, pp. 168–173.

[10] R. Svenningsson, J. Vinter, H. Eriksson, and M. Törngren, "Modifi: a model-implemented fault injection tool," in *International Conference on Computer Safety, Reliability, and Security*, E. Schoitsch, Ed. Berlin, Heidelberg: Springer, 2010, pp. 210–222.

[11] M. Saraoğlu, A. Morozov, M. T. Söylemez, and K. Janschek, "Errorsim: A tool for error propagation analysis of simulink models," in *International Conference on Computer Safety, Reliability, and Security*. Cham: Springer, 2017, pp. 245–254.

[12] T. Fabarisov, I. Mamaev, A. Morozov, and K. Janschek, "Model-based fault injection experiments for the safety analysis of exoskeleton system," in *Proceedings of the 30th European Safety and Reliability Conference and the 15th Probabilistic Safety Assessment and Management Conference*, 2020. [Online]. Available: https://arxiv.org/abs/2101.01283

[13] M. Moradi, B. V. Acker, K. Vanherpen, and J. Denil, "Model-implemented hybrid fault injection for simulink (tool demonstrations)," in *Cyber Physical Systems. Model-Based Design - 8th International Workshop, CyPhy 2018, and 14th International Workshop, WESE 2018, Turin, Italy, October 4-5, 2018, Revised Selected Papers*, ser. Lecture Notes in Computer Science, R. D. Chamberlain, W. Taha, and M. Törngren, Eds., vol. 11615. Springer, 2018, pp. 71–90. [Online]. Available: https://doi.org/10.1007/978-3-030-23703-5_4

[14] A. Ramírez Bárcenas, M. García-Valderas, and C. López-Ongil, "Fault injection at model level for signal data processing in a multisensor instrument for planetary exploration," *IEEE Sensors Journal*, vol. 24, no. 24, pp. 41 277–41 287, 2024.

[15] MathWorks, "Simulink fault analyzer," 2025. [Online]. Available: https://in.mathworks.com/products/simulink-fault-analyzer.html

[16] J. Zhang, D. Ghobari, M. Sabetzadeh, and S. Nejati, "Simulink mutation testing using codebert," *arXiv preprint arXiv:2501.07553*, 2025.

[17] H. I. Ceylan, O. Kilinççeker, M. Beyazit, and S. Demeyer, "MUT4SLX: fast mutant generation for simulink," in *38th IEEE/ACM International Conference on Automated Software Engineering, ASE 2023, Luxembourg, September 11-15, 2023*. IEEE, 2023, pp. 2086–2089. [Online]. Available: https://doi.org/10.1109/ASE56229.2023.00093

[18] S. Nuyens, H. I. Ceylan, O. Kilinççeker, M. Beyazit, and S. Demeyer, "MUT4SLX: extensions for mutation testing of stateflow models," in *IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2024 - Companion, Rovaniemi, Finland, March 12, 2024*. IEEE, 2024, pp. 215–218. [Online]. Available: https://doi.org/10.1109/SANER-C62648.2024.00035

[19] B. Liu, S. Nejati, Lucia, and L. C. Briand, "Effective fault localization of automotive simulink models: achieving the trade-off between test oracle effort and fault localization accuracy," *Empir. Softw. Eng.*, vol. 24, no. 1, pp. 444–490, 2019. [Online]. Available: https://doi.org/10.1007/s10664-018-9611-z

[20] K. Chen, M. Wen, H. Jia, R. Wu, and H. Jin, "Towards effective and efficient error handling code fuzzing based on software fault injection," in *IEEE International Conference on*

Software Analysis, Evolution and Reengineering, SANER 2024, Rovaniemi, Finland, March 12-15, 2024. IEEE, 2024, pp. 320–331. [Online]. Available: https://doi.org/10.1109/SANER60148.2024.00039

[21] E. Bartocci, L. Mariani, D. Nickovic, and D. Yadav, "Signal feature coverage and testing for cps dataflow models," *ACM Trans. Softw. Eng. Methodol.*, vol. 34, no. 7, Aug. 2025. [Online]. Available: https://doi.org/10.1145/3714467

[22] Q. Wang, J. Rios, S. Jha, K. Shanmugam, F. Bagehorn, X. Yang, R. Filepp, N. Abe, and L. Shwartz, "Fault injection based interventional causal learning for distributed applications," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, no. 13, pp. 15 738–15 744, Jul. 2024. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/26868

[23] Y.-J. Zhang and L.-S. Hu, "Fault propagation inference based on a graph neural network for steam turbine systems," *Energies*, vol. 14, no. 2, p. 309, 2021.

[24] Mathworks. (2025) Design a guidance system in matlab and simulink. Mathworks. [Online]. Available: https://nl.mathworks.com/help/simulink/slref/designing-a-guidance-system-in-matlab-and-simulink.html

[25] Mathworks. (2025) Model an anti-lock braking system. Mathworks. [Online]. Available: https://nl.mathworks.com/help/simulink/slref/modeling-an-anti-lock-braking-system.html

[26] L. Briand, D. Bianculli, S. Nejati, F. Pastore, and M. Sabetzadeh, "The case for context-driven software engineering research: Generalizability is overrated," *IEEE Software*, vol. 34, no. 5, pp. 72–75, 2017.