Machine Learning-assisted Fault Injection

Mehrdad Moradi, Bentley James Oakes, and Joachim Denil
University of Antwerp and Flanders Make vzw, Belgium
{Mehrdad.Moradi, Bentley.Oakes, Joachim.Denil}@uantwerpen.be

***Abstract*—Fault Injection (FI) is a method for system validation and verification in which the tester evaluates the system behavior resulting from the introduction of faults into the system under test. This paper proposes a model-based approach to improve the efficiency of the FI process by utilizing Machine Learning (ML) and formalized domain knowledge. This ML algorithm uses a probabilistic automaton to reduce the manual effort required in the testing procedure as the algorithm can automatically make decisions and predictions about catastrophic fault parameters. This assists the tester in dealing with complicated and broad-scale systems by enabling higher fault coverage with fewer simulations.**

***Index Terms*—Fault injection, Machine learning, Validation and verification, Safety assessment, Domain knowledge, Bayesian networks**

I. Introduction

Cyber-physical systems (CPS) are complex engineered systems that have tight integration between the cyber (computation and networking) and physical components [1]. An example of a CPS is an autonomous vehicle, which will play an increasing role in daily life within a few years. These systems are safety-critical as faults in the system can result in catastrophic situations resulting in severe injury or loss of life. For example, faulty sensor readings in an autonomous car might result in the non-detection of a pedestrian that is crossing the street, with life-threatening consequences.

Vendors may perform safety assessment by executing Fault Injection (FI) experiments. FI is a testing method aiming to stress the real or virtual system to validate, verify, and evaluate the safety mechanisms of the systems [2]–[4]. In FI, a fault is introduced into the system, such that the system's behavior can then be analyzed to evaluate whether or not this fault results in unsatisfactory or dangerous system performance.

Faults have three main dimensions [3] which creates the *fault space*: (i) *type*: what should be injected? (ii) *location*: where should it be injected? (iii) *time*: when should it be activated? For modern CPSs, choosing the fault with the most impact from such a large fault space requires an enormous amount of time and effort. Hence, traditional FI methodologies are ineffective for current complex systems.

Research problem: FI is traditionally based on experimentation and simulation on hardware and/or software prototypes of a system [2], [5]. The state-of-the-art techniques for FI are mainly aimed at the evaluation of the computational hardware and software in the final phases of system development [6]–[9]. This approach is very late in the design cycle, which increases the cost for vendors to find a fault and modify the system and repair it. Traditional FI methods are also mainly random-based or user-oriented approaches that are time-consuming and computationally expensive. This is due to the number of possible configurations for a FI experiment, potentially resulting in an infinite configuration space. Therefore, the first issue in FI is to select those critical faults (and their configuration of location, amplitude, and time), which are most likely to lead to the system's failure. This set of critical faults can then be injected and analyzed to determine the most significant risks to system behavior.

In our view, the enormous complexity of CPS combined with the (almost) infinite configuration space of faults demands an automated and optimized approach to FI. Manual approaches are too cumbersome and restricted in scope to accurately locate and reason about catastrophic situations in today's CPS. Therefore, we propose here a new direction of utilizing state-of-the-art Machine Learning (ML) techniques to perform model-implemented FI.

Proposed method: Our research aims to tackle the system's complexity with less cost by focusing on model-based FI in the design phase of the system. Our technique works on the three main fault's parameters (type, time, and location) to choose a combination of them that can cause the system to fail.

In the proposed approach, the ML algorithm controls the parameters of the faults. It first uses the domain knowledge provided by the tester to initialize the test procedure. Domain knowledge is a simplified model of the model under test with some defined rules or constraints regard to the model. This domain knowledge is modeled as a *Bayesian Network* (BN). A BN is a probabilistic automaton consisting of states and transitions which represent the system behavior and relation between different system components. The ML algorithm first loads the BN with domain knowledge, then it trains the BN by running simulations, as there is not yet a static database when the system is in model level. The ML algorithm then uses the underlying BN to predict the catastrophic faults that fail the system [10]. Next, FI aims the ML algorithm to simulate and modify its prediction.

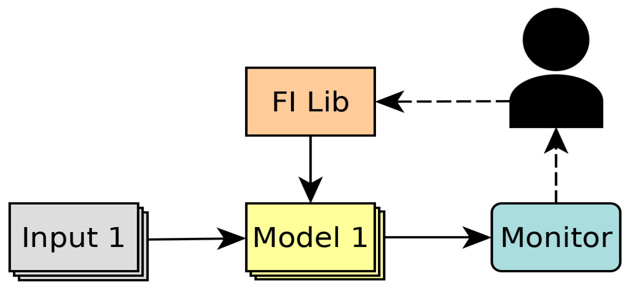
The remainder of this paper is organized as follows. In section II, we explain the proposed technique and its main parts. Next, we illustrate how we will validate the technique in section III. Then, we describe the benefits and limitations of our technique in section IV, and finally, we conclude the paper in section V.

II. Technique

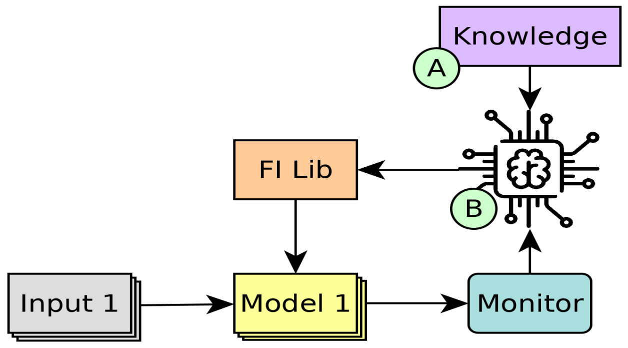
Our recent work has demonstrated the feasibility of applying *reinforcement learning* (RL) for optimizing FI [11]. In that paper, we configured the RL algorithm to explore the fault space efficiently. The RL algorithm searches for the critical faults in the fault space, which violate the safety requirement. We defined the safety requirement to be used as a reward function for the RL agent. We also compared our method with Monte-Carlo FI (MCFI), and the proposed technique was more efficient in terms of fault coverage (the ratio of the number of catastrophic faults to the total number of injected faults) [3] and the number of required simulations to find the first critical fault.

In Fig. 1, we illustrate our proposed approach of model-implemented FI against traditional approach [12]. In the traditional approach in Fig. 1a, the tester selects a group of fault parameters based on their own experience and the faults in the fault library. Then, the user creates a group of faulty models and runs the simulation. Next, the user investigates the results and manually chooses the next group of faults.

In this paper, we propose the architecture in Fig. 1b. Two components are added to the traditional approach to optimize model-implemented FI: (i) explicit domain knowledge (marked A), and (ii) a predictor/decision-maker (marked B). The proposed approach tries to find critical faults to increase the fault coverage and decrease the run time of the experiment. The user provides domain knowledge, and the ML algorithm uses that knowledge to initialize the test setup, then the ML algorithm tries to predict the critical fault in each simulation. The tester role is thereby eliminated by combining the explicit modeling of domain knowledge and the ML algorithms.



(a) Traditional method.



(b) Proposed method.

Fig. 1: Comparing fault injection in industrial practice to our proposed approach.

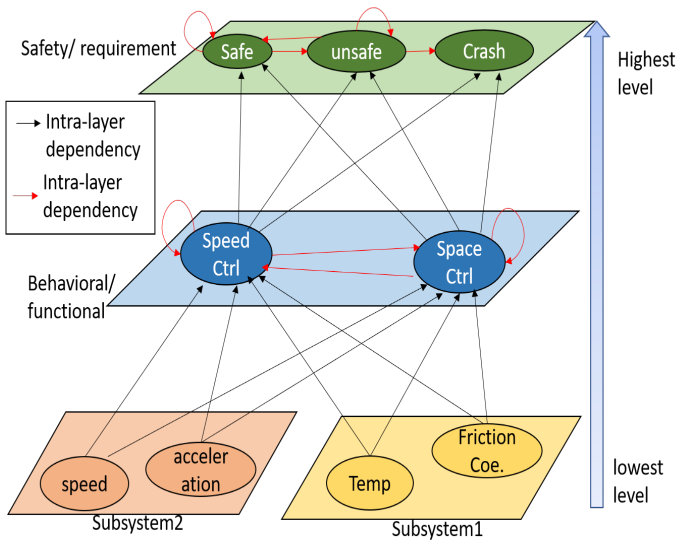


Fig. 2: Bayesian network of the model under test at multiple abstraction levels.

A. Domain Knowledge

Domain knowledge is the term for a simplified version of the model under test which the ML algorithm can interact with easily. This knowledge assists the ML algorithm in predicting the critical fault parameter faster and more accurately [13]. Domain knowledge includes the requirements of the system, as well as its modes, architecture, constraints, and assumptions. For example, this can include ranges of input/output or information about the (dis-)continuity of system operation modes. The domain expert could model this knowledge as a Probabilistic Automaton (PA). PA is based on states and transitions with corresponding probability. An example of such an automaton is Bayesian Networks (BN), which represents the conditional dependency between a set of variables. In our proposed approach, the domain expert creates an initial BN with states and probabilities on each transition. Then, the probability of each state will automatically be updated by the ML algorithm during the continual learning steps.

In Fig. 2 we illustrate the BN for Adaptive Cruise Control (ACC). ACC is a driving assistant system in a modern car that controls the acceleration to satisfy a predefined speed and safe distance. Hence, the ACC is safety critical at a high level of abstraction, the *safety/requirement* model. Second is the *behavioral/functional* model, and then the detailed *model variables and parameters* at the lowest level of abstraction. Each state connects to relevant states in both the lower and upper levels, and in some levels are dependent on each other.

B. Machine Learning Algorithm

In this paper, we use a Dynamic Bayesian Network (DBN) [14], specifically a Temporal Bayesian Network (TBN). The main idea of a DBN is that we have a static BN for each point in time, and there is a temporal link in each state between

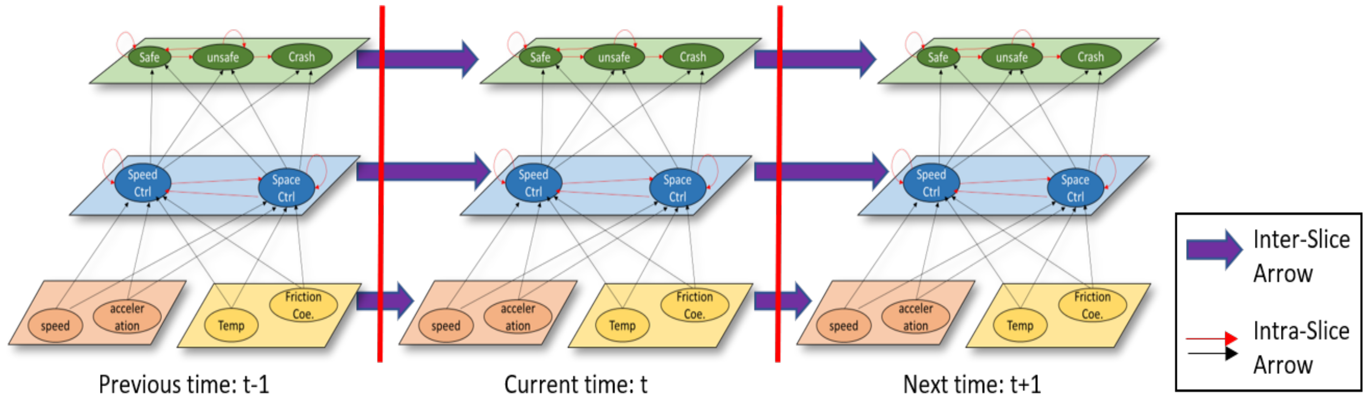


Fig. 3: 3-Temporal Bayesian network of ACC.

the current step and the previous step. Fig. 3 presents a 3-TBN representing states of ACC in the present time, previous step, and the next step. Structure of BN in each time step are identical, but probabilities are different. The 3-TBN is the core of our proposed ML algorithm, which must first be trained and then becomes a data repository to predict the fault parameters and make decisions.

1) Continual learning: After defining the domain knowledge, we have states and transitions at multiple levels of abstraction. However, we require probability values for each state. For this purpose, we must train the 3-TBN with the simulation as, at this point in the design cycle, there is no static database. The usage of multiple levels of abstractions in domain knowledge and simulation aid the training to tackle data scarcity. The learning process updates the probabilities on the 3-TBN transitions while performing simulations. The application of different input scenarios to the system under test results in a variety of system action modes and states. Through exposure to a variety of system behaviors, the 3-TBN is trained by having all transition probabilities converge to specific values.

2) Prediction and decision-making: After training, the 3-TBN acts as a *predictor* or *decision-maker*. It predicts the next state of the systems, and based on that prediction, it decides whether to inject a fault or not. It also determines the parameters of the critical fault such that the FI block in the Fig.1b can perform the actual FI. For predicting the parameters for critical faults we utilize *conditional probability*, as specified for two events A and B in Eq. (1).

$$P(A|B) = P(B|A) \times P(A)/P(B) \qquad (1)$$

This equation is applied to each safety requirement to calculate the probability of satisfying the requirement concerning other parameters at each point in time. Hence, the hazardous situations are identified by obtaining information about which specifications are in hazard, and the time when it most likely occurs. This determines the critical fault's activation time and location, while the fault's amplitude is based on the parameter ranges derived from the domain knowledge provided by the domain expert.

Also, the learning algorithm learns from its injection experience. For example, if the injection causes a specification to almost fail, but it does not yet create a hazard, the ML algorithm will magnify the fault amplitude in the next simulation and try to inject the fault again.

This approach, therefore, injects a fault and runs simulations to monitor the system's requirements. Then, the ML algorithm measures the severity of violating system's specification and modifies the fault parameters in the next iteration to find more severe faults. The ML algorithm iterates on these steps until discovering a satisfactory set of critical faults for further use in the safety assessment process.

C. Corner Case Input Scenarios

For having a complete testing cycle, we need to examine as many input scenarios as possible. Since applying all possible input cases is infeasible for complex systems, we need to explore the input test cases and find hazardous corner cases that have a higher chance of creating a hazard. This search is performed using the same procedure as described above. In the domain knowledge, we take behavior of other agents into account, and with considering kinematic equations, we can omit irrelevant input cases and focus on the most critical inputs. For example, in the ACC use case, if the initial position of two cars is close and the velocity of the rear car is higher than the car ahead, the chance of an accident is higher. Therefore, we select this scenario as a high priority for simulation and experimentation.

III. Validation

We will validate the technique by comparing it with state-of-the-art methods. Statistical FI is widely used in hardware-based FI [15] and software-based FI [16], [17], and also has significant results compared to random-based methods. There are also studies in which authors exploit the ML-based approaches to increase the efficiency of their tests [18], [19]. These techniques will be implemented alongside our own in a common use case, such that we can compare the effectiveness of our method.

IV. Discussion

This section briefly discusses the strengths and limitations of our approach.

Benefits: This approach uses a model-implemented approach that can be applied to an earlier phase of system development than traditional techniques. Also, it can readily be applied to heterogeneous and large scale CPS with high levels of complexity. This is due to the use of multiple levels of abstraction and a BN as the main source of analysis, which has a low computational cost.

The approach also explores and prunes the fault space based on a probability theorem instead of random-based approaches, ensuring that valid and plausible faults are obtained. It tries to find potential hazards at each point in time and discovers catastrophic fault parameters.

In this approach, we also investigate the corner case input scenarios, as this has a profound effect on system verification and validation. Using BN for both FI and finding corner cases enables the tester to investigate the model under test with a high confidence of accuracy.

The test engineer can also manage trade-offs between performance and fault coverage. For example, constraints on the FI approach can be set in a way that the ML algorithm only investigates faults with the highest probability (possible catastrophic faults) for faster experimentation. Alternatively, the tester can also configure the ML algorithm to explore an expanded space of possible faults, which increases the run time of the simulation compared with a limited exploration experiment but would provide higher fault coverage.

Limitations: One limitation of the proposed technique is that the tester must have sufficient knowledge about the model under test. This knowledge enables the framework to explore the model efficiently, but it is unclear how to decide when knowledge is missing.

The tester must also properly model the BN. The BN structure has a profound effect on the simulation result because it acts as a database for the ML algorithm. With a naive BN, the proposed method may not obtain acceptable performance and efficiency. Therefore, future work will define a systematic way to build up an appropriate BN from domain knowledge.

V. Conclusion

This paper proposes a new direction to parameter setting for fault injection, which the machine learning algorithm and domain knowledge optimize the injection process. This approach improves the fault coverage and performance of testing and produces a valid result. The effort required by the test engineer is also reduced, decreasing the possibility of human error and speeding up the system validation and verification process. We also investigate the input corner cases, ensuring a full test cycle for validation, verification, and safety assessment of the model under test.

Acknowledgment

This work was partly funded by Flanders Make vzw, the strategic research centre for the Flemish manufacturing industry; and by the Flanders Make project aSET (grant no. HBC.2017.0389) of the Flanders Innovation and Entrepreneurship agency (VLAIO). The intelligence icon in Fig. 1b is made by Freepik from www.flaticon.com.

References

[1] E. A. Lee, "Cyber physical systems: Design challenges," in *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*. IEEE, 2008, pp. 363–369.

[2] M.-C. Hsueh, T. K. Tsai, and R. K. Iyer, "Fault injection techniques and tools," *Computer*, vol. 30, no. 4, pp. 75–82, 1997.

[3] A. Benso and P. Prinetto, *Fault injection techniques and tools for embedded systems reliability evaluation*. Springer Science & Business Media, 2003, vol. 23.

[4] M. Le and Y. Tamir, "Fault injection in virtualized systems—challenges and applications," *IEEE Transactions on Dependable and Secure Computing*, vol. 12, no. 3, pp. 284–297, 2014.

[5] J. Arlat *et al.*, "Fault injection for dependability validation: a methodology and some applications," *IEEE Transactions on Software Engineering*, vol. 16, no. 2, pp. 166–182, Feb 1990.

[6] J. Carreira, H. Madeira, and J. G. Silva, "Xception: A technique for the experimental evaluation of dependability in modern computers," *IEEE Transactions on Software Engineering*, vol. 24, no. 2, pp. 125–136, 1998.

[7] S. Han, K. G. Shin, and H. A. Rosenberg, "Doctor: An integrated software fault injection environment for distributed real-time systems," in *Proceedings of 1995 IEEE International Computer Performance and Dependability Symposium*. IEEE, 1995, pp. 204–213.

[8] G. A. Kanawati, N. A. Kanawati, and J. A. Abraham, "Ferrari: A flexible software-based fault and error injection system," *IEEE Transactions on computers*, vol. 44, no. 2, pp. 248–260, 1995.

[9] W.-L. Kao and R. K. Iyer, "Define: A distributed fault injection and monitoring environment," in *Proceedings of IEEE Workshop on Fault-Tolerant Parallel and Distributed Systems*. IEEE, 1996, pp. 252–259.

[10] J. Pearl, *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Elsevier, 2014.

[11] M. Moradi, B. J. Oakes, M. Saraoğlu, A. Morozov, K. Janschek, and J. Denil, "Exploring fault parameter space using reinforcement learning-based fault injection a preprint," *In Safety and Security of Intelligent Vehicles (SSIV)*, 2020.

[12] M. Moradi, B. Van Acker, K. Vanherpen, and J. Denil, "Model-implemented hybrid fault injection for Simulink (tool demonstrations)," in *Cyber Physical Systems. Model-Based Design*, R. Chamberlain, W. Taha, and M. Törngren, Eds. Cham: Springer International Publishing, 2019, pp. 71–90.

[13] S. Jha, S. Banerjee, T. Tsai, S. K. Hari, M. B. Sullivan, Z. T. Kalbarczyk, S. W. Keckler, and R. K. Iyer, "Ml-based fault injection for autonomous vehicles: A case for Bayesian fault injection," in *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, June 2019, pp. 112–124.

[14] D. Koller, J. Weber, T. Huang, J. Malik, G. Ogasawara, B. Rao, and S. Russell, "Towards robust automatic traffic scene analysis in real-time," in *Proceedings of 12th International Conference on Pattern Recognition*, vol. 1. IEEE, 1994, pp. 126–131.

[15] I. Tuzov, D. de Andrés, and J.-C. Ruiz, "Accurate robustness assessment of hdl models through iterative statistical fault injection," in *2018 14th European Dependable Computing Conference (EDCC)*. IEEE, 2018, pp. 1–8.

[16] S. K. S. Hari, S. V. Adve, H. Naeimi, and P. Ramachandran, "Relyzer: Exploiting application-level fault equivalence to analyze application resiliency to transient faults," *ACM SIGARCH Computer Architecture News*, vol. 40, no. 1, pp. 123–134, 2012.

[17] N. Tian, D. Saab, and J. A. Abraham, "Esift: Efficient system for error injection," in *2018 IEEE 24th International Symposium on On-Line Testing And Robust System Design (IOLTS)*, July 2018, pp. 201–206.

[18] F. R. da Rosa, R. Garibotti, L. Ost, and R. Reis, "Using machine learning techniques to evaluate multicore soft error reliability," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 66, no. 6, pp. 2151–2164, June 2019.

[19] H. Khosrowjerdi, K. Meinke, and A. Rasmusson, "Virtualized-fault injection testing: A machine learning approach," in *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 2018, pp. 297–308.