Tutorial: LLTFI and the Art of Fault Injection

Karthik Pattabiraman, Abraham Chan
The University of British Columbia (UBC), Vancouver, BC, Canada
Email: karthikp@ece.ubc.ca, abrahamc@ece.ubc.ca

***Abstract*—Fault injection has been a well-researched area in the dependable and reliable systems community. Nevertheless, a simple framework for fault injection that combines both software and hardware errors in an adaptable way, has not previously existed before the debut of LLFI, an LLVM-based fault injector tool. We present a tutorial for LLTFI, an fault injector tool extending LLFI, which is capable of injecting faults into both machine learning and native code applications. Assuming some prior knowledge of fault injection, this tutorial covers the fundamentals, design philosophy, and application of LLTFI.**

***Index Terms*—Fault injection, Error resilience, Machine learning**

I. Tutorial Description

Fault injection (FI) has been extensively researched in the DSN community [1–4]. However, there is no easy framework allowing ordinary developers and testers to inject both hardware and software faults, in a customizable fashion. At the University of British Columbia (UBC), our group, the Dependable Systems Lab, has been developing one such framework called LLTFI[1] [5], which is the successor of another previously developed tool in our group for more than a decade called LLFI[2] [6, 7], which is widely used in academia and industry. LLTFI, which stands for Low-Level Tensor Fault Injector, augments LLFI with the capability to inject faults into Machine Learning (ML) applications written using a wide-variety of frameworks, such as TensorFlow [8] or PyTorch [9], as well as native code. LLTFI is based on the popular open-source LLVM compiler, and allows fault injection to be combined with program analysis techniques. We have used LLTFI to perform fault injection evaluations of computer vision ML models [5] and large language models (LLMs) [10].

This tutorial will demonstrate the basics of LLTFI, the design philosophy and its use. The tutorial does not assume any prior knowledge of LLVM, but (some) prior knowledge of FI will be assumed. We will start by showing how to perform simple fault injection (FI) experiments for both regular (i.e., C / C++) programs and ML programs (i.e., TensorFlow and PyTorch models). Then, we will explore how to interpret the results of FI experiments and use them in real-world case studies. Finally, we will delve into the internals of LLTFI and learn how to write new fault injectors using the APIs of the framework, and to extend it. We will also discuss open issues in the research area of FI.

[1]https://github.com/DependableSystemsLab/LLTFI
[2]https://github.com/DependableSystemsLab/LLFI

II. Goal

This tutorial will teach the audience the basics of LLTFI, the design philosophy and how to use it. We will start by learning to perform simple fault injection experiments for both regular programs and ML programs. We will also explore how to interpret the results of the fault injection experiments and use them in real-world case studies. Finally, we will delve into the internals of LLTFI and learn how to write new fault injectors using the APIs of the framework, and to extend it.

III. Target Audience

The tutorial targets two groups. The first group is developers and testers who use or plan to use fault injection (FI) in their day to day activities. A number of hands-on activities and exercises will be used throughout the tutorial to help this group apply FI to different applications. The second group are researchers who want to use LLTFI (or similar tools) in their research. The tutorial does not assume any prior knowledge of LLVM, but (some) prior knowledge of FI will be assumed. Though the tutorial will focus on LLTFI, the principles discussed can be broadly applicable to other (software-based) fault injection tools. We will also discuss open issues in the research area of FI.

IV. Tutorial Outline

1) Introduction to FI, Philosophy of LLTFI, LLTFI features and high-level overview (30 minutes)
2) Two different case studies for illustrating the use of LLTFI, one injecting faults into regular applications, and the other for injecting faults into ML applications (60 minutes)
3) Internals of LLTFI and writing new fault injectors, both in regular and ML programs (60 minutes)
4) Open issues and research directions in fault injection, free-form discussion (30 minutes)

V. Takeaways

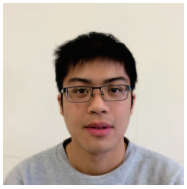
The following are the main intended takeaways from the tutorial:

1) Understanding of the design of LLTFI and its capabilities
2) Ability to use LLTFI to inject faults into both regular and ML programs, and results analysis
3) Ability to write new fault injectors in the LLTFI framework and to extend its capabilities
4) Knowledge of the open issues and research directions in the area of fault injection tools

2833-292X/24/$31.00 ©2024 IEEE
DOI 10.1109/DSN-S60304.2024.00030

VI. Biographies of Presenters

Karthik Pattabiraman is a Professor of Electrical and Computer Engineering (ECE) at the University of British Columbia (UBC) in Canada. He received his PhD and MS in 2009 and 2004 in Computer Science from the University of Illinois at Urbana-Champaign (UIUC). Before joining UBC in 2010, he was a postdoctoral researcher at Microsoft Research (MSR), Redmond. Karthik's research interests are in dependable systems, cyber-physical systems, and software security. In particular, his group has developed a number of techniques to tolerate hardware faults using software, and also perform fault injection in the software stack.



Abraham Chan is a PhD candidate in Computer Engineering at the University of British Columbia (UBC), advised by Professors Karthik Pattabiraman and Sathish Gopalakrishnan. His primary research interests are in ML reliability, with the goal of developing resilient ML systems against faulty training data in safety-critical applications. His secondary research explores the construction of reliable ML models against soft errors affecting hardware during inference. Prior to starting his PhD, he worked as a compiler engineer at Huawei, where he worked on compiler optimizations for AI accelerators and mobile graphics.

References

[1] R. Alexandersson and J. Karlsson, "Fault injection-based assessment of aspect-oriented implementation of fault tolerance," in *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2011.

[2] A. Thomas and K. Pattabiraman, "Error Detector Placement for Soft Computing Applications," in *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2013.

[3] S. Jha *et al.*, "ML-based Fault Injection for Autonomous Vehicles: A Case for Bayesian Fault Injection," in *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2019.

[4] A. Mahmoud *et al.*, "PyTorchFI: A Runtime Perturbation Tool for DNNs," in *IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, 2020.

[5] U. Agarwal, A. Chan, and K. Pattabiraman, "LLTFI: Framework Agnostic Fault Injection for Machine Learning Applications," in *IEEE International Symposium on Software Reliability Engineering (ISSRE)*, 2022.

[6] Q. Lu, M. Farahani, J. Wei, A. Thomas, and K. Pattabiraman, "LLFI: An Intermediate Code-Level Fault Injection Tool for Hardware Faults," in *IEEE International Conference on Software Quality, Reliability and Security (QRS)*, 2015.

[7] J. Wei, A. Thomas, G. Li, and K. Pattabiraman, "Quantifying the Accuracy of High-Level Fault Injection Techniques for Hardware Faults," in *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2014.

[8] M. Abadi *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. [Online]. Available: https://www.tensorflow.org/

[9] A. Paszke *et al.*, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems 32*, 2019.

[10] U. Agarwal, A. Chan, and K. Pattabiraman, "Resilience Assessment of Large Language Models under Transient Hardware Faults," in *IEEE International Symposium on Software Reliability Engineering (ISSRE)*, 2023.