Fuzzing to Identify Undiscovered Bugs in Scientific Software

Raunak Shakya[1], Jonathan Gibson[1], and John Brackins[2] (Advisor: Dr. Akond Rahman[1])
[1]Department of Computer Science, Tennessee Tech University
[2]Department of Civil and Environmental Engineering, Tennessee Tech University

rshakya@students.tntech.edu
jagibson44@students.tntech.edu
jtbrackins42@students.tntech.edu

Background

- **Scientific software**: software applications **primarily focused on exploration and analysis of data**
- **Mostly developed by researchers/graduate students with deadlines**, not software developers; potential introduction of **bugs**

Motivation

- As researchers write more code, higher probability of errors follow (Soegler, 2015)
- Traditional software testing methods are **not always enough to find critical, result-altering bugs (*e.g.* Bhandari Neupane et al. 2019)**

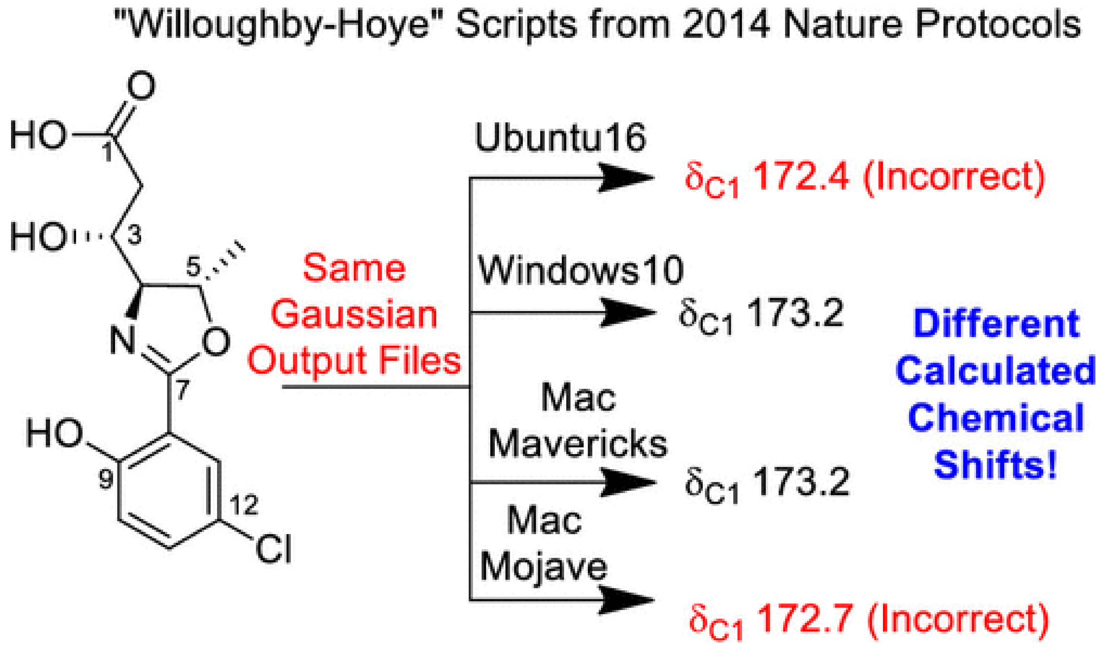


Figure 1: Result-altering bugs due to different operating systems, reproduced from Bhandari Neupane et al. (2019).

- Need for a "new" software testing technique, **fuzzing: feeding predesigned data to a program to trigger crash/unexpected behavior**

Research Goal

The goal of this research project is to help scientists in conducting correct and verifiable scientific computations by **identifying latent bugs in scientific software**.

Research Question

How can we **identify crash-prone inputs** that can be used as fuzz data **to discover bugs** in scientific software?

Hypothesis

Through qualitative analysis we can identify **characteristics of bugs** in scientific software that we can leverage to **identify undiscovered bugs**.

Datasets

List of fuzzed Julia packages:

- FFTW.jl
- GLFW.jl
- HTTP.jl
- WebSockets.jl
- SymPy.jl
- LightXML.jl
- LinearOperators.jl

Methodology

- Traditional fuzzing: **hand-written fuzzers**
- Three typical methods:

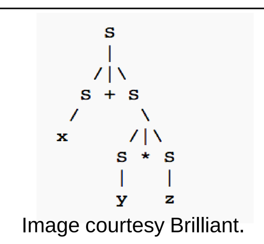
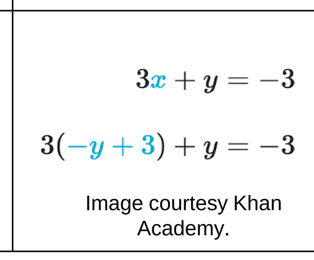
True random inputs		• **Purely random** strings • Less likely to find bugs
Generation-based		• Use **grammars** to generate inputs • More likely to find bugs
Mutation-based		• **Modify existing** inputs • More likely to find bugs

Table 1: Three approaches for traditional fuzzing.

- We first implemented traditional hand-written fuzzers in Python for 7 Julia repositories
- Combinations of random, generation-based, and mutation-based fuzzers

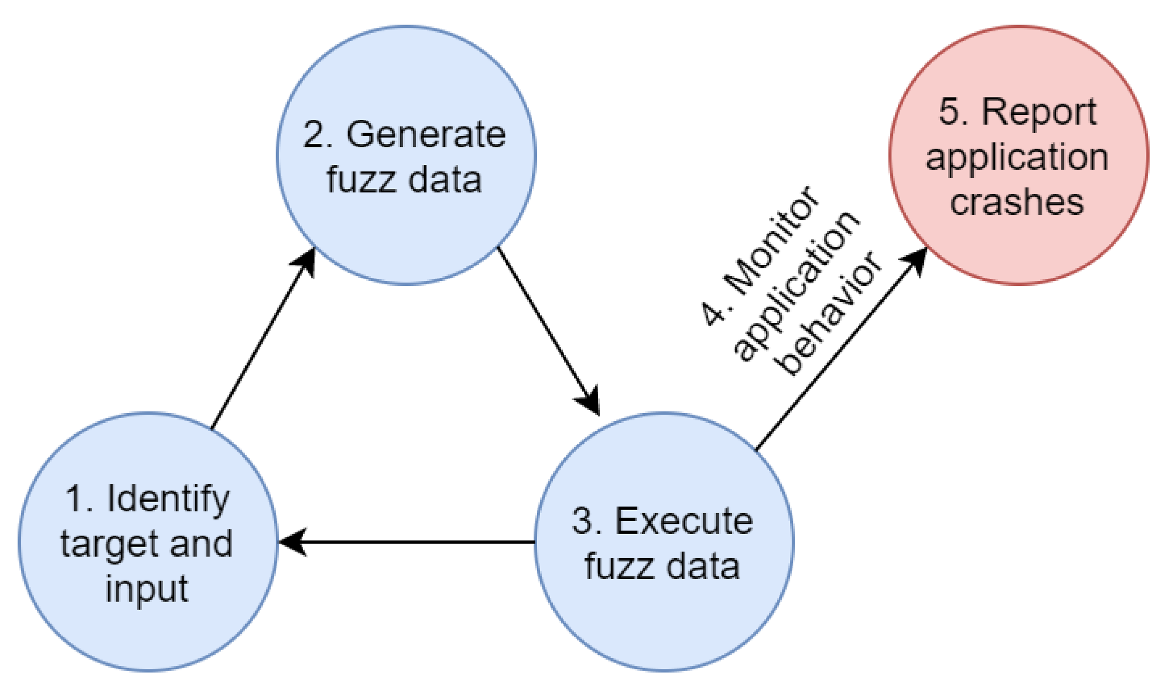


Figure 2: Different stages of fuzzing a program..

- **Wrote traditional fuzzers for each unique function** in each of 7 Julia libraries
- We then began to implement **automated fuzzing techniques** utilizing **machine learning following Cummins et al. (2018)**
  - Mined 8,211 public Julia repositories from GitHub (**107K Julia files**, in 34K directories, **~8.2M lines of code**) to obtain training **corpus**
  - Applied consistent variable naming/code formatting to ease machine learning process
  - Will use Long Short-Term Memory (**LSTM**) style of Recurrent Neural Network to learn structure and syntax of Julia
- Sample LSTM to create very large (~1M files) synthetic corpus as a test suite: represents **"repositories that have yet to be written"**

Methodology (continued)

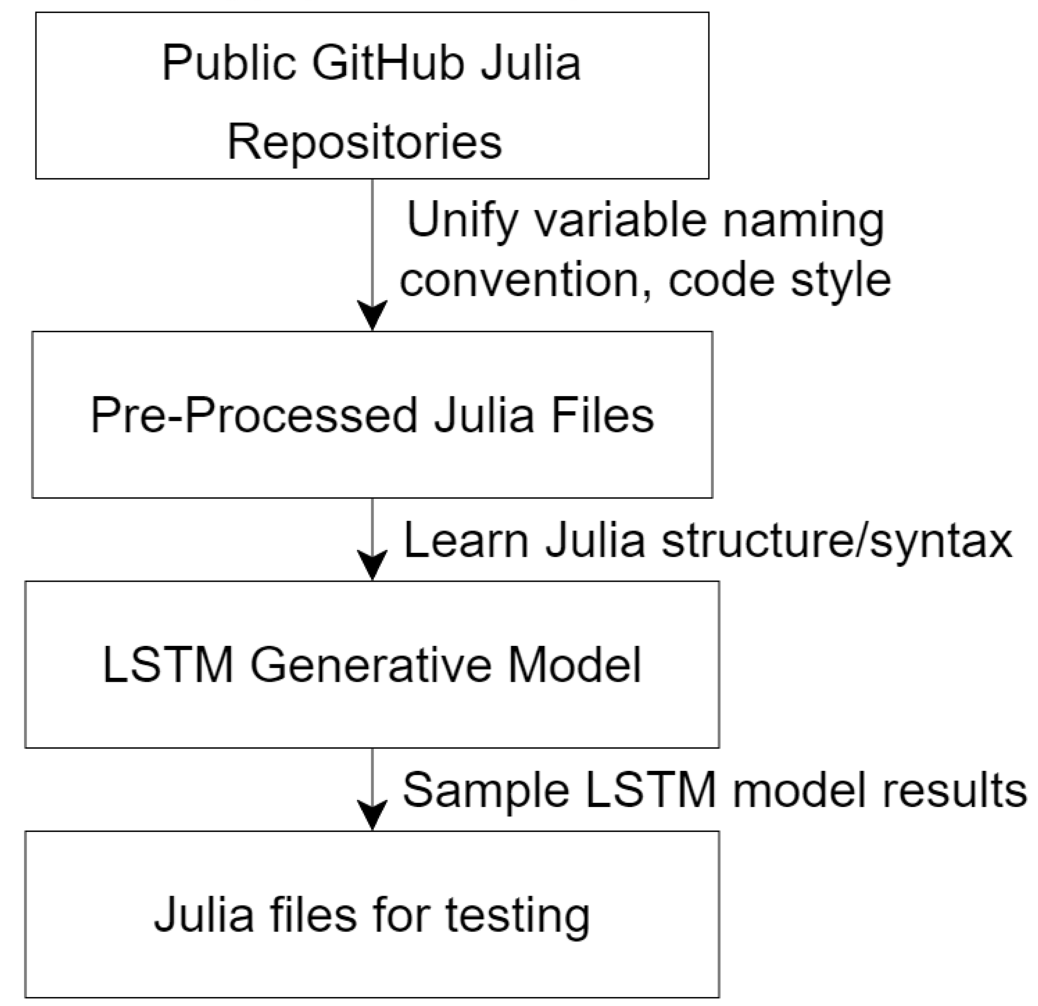


Figure 3: Machine learning workflow to generate test suite.

- Create multiple testbeds to run same test files; use **differential testing** of code outputs to reveal bugs

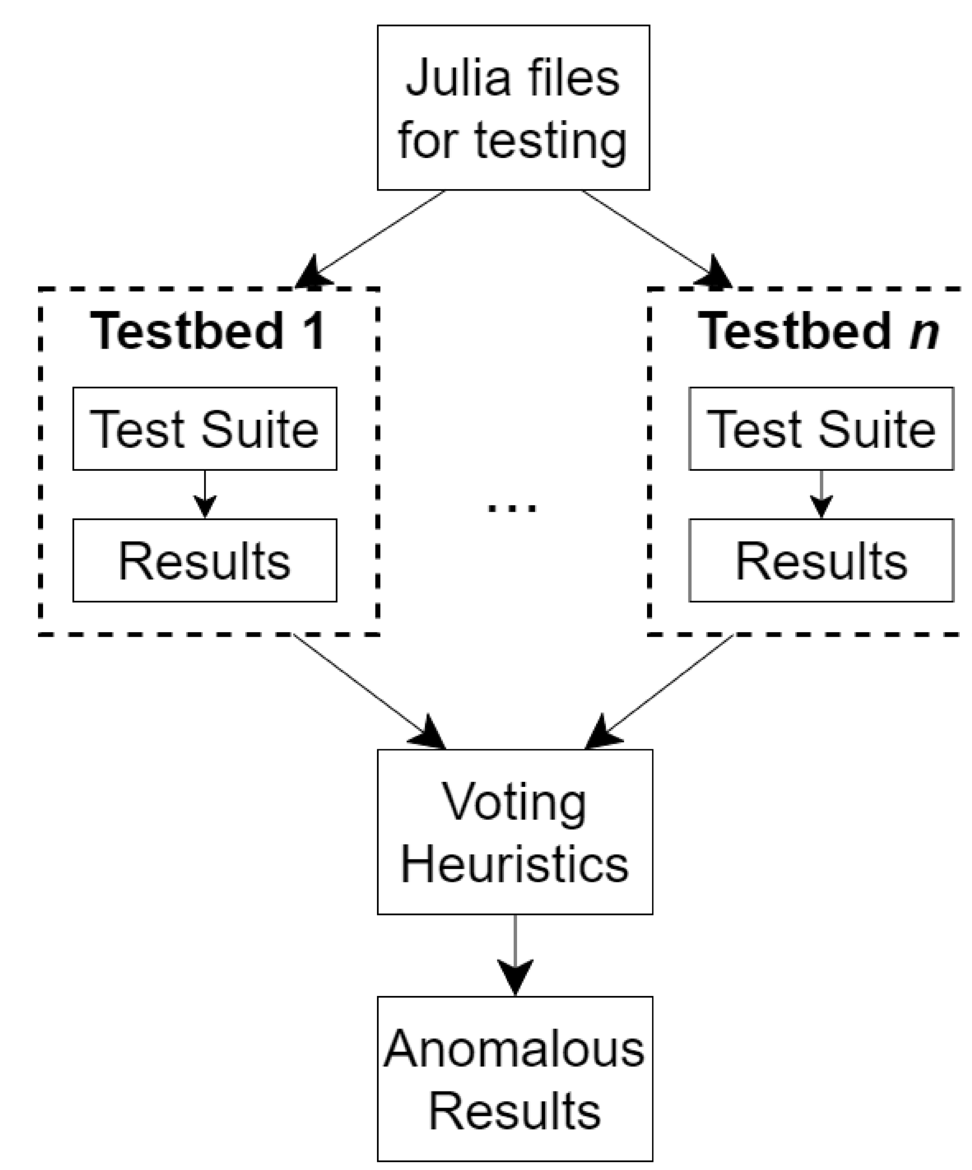


Figure 4: Testbed approach to detect anomalous results, adapted from Cummins et al. (2018).

- Under further study of suspect scripts, **potential bugs** can be **readily isolated** and addressed

Results

- Traditional fuzzing has been completed for 7 Julia repositories using Python
- Traditional approach required **over 3 months** and does not scale well; **a scalable solution is needed**

Results (continued)

- **Nevertheless, preliminary analysis using traditional techniques found several *unhandled* exception conditions** in the 7 Julia libraries, for a total of 9 bugs found:

FFTW Bugs	HTTP Bugs
MethodError	MethodError
Invalid escape sequence	Invalid escape sequence
String juxtapose error	Invalid string syntax error
UndefVarError	BoundsError
	StatusError

Table 2: Bugs discovered using traditional fuzzing approach.

Conclusions and Future Work

- The present work shows the inefficiency of traditional fuzzing methods and demonstrates potential for machine learning techniques to improve fuzzing efficiency
- **Present methodology using machine learning techniques can be extended to a much larger set of open-source repositories** to evaluate their functionality and help reduce the presence of defects

References

- Bhandari Neupane, J., Neupane, R. P., Luo, Y., Yoshida, W. Y., Sun, R., & Williams, P. G. (2019). Characterization of leptazolines A–D, polar oxazolines from the cyanobacterium Leptolyngbya sp., reveals a glitch with the "Willoughby–Hoye" scripts for calculating NMR chemical shifts. Organic letters, 21(20), 8449-8453.
- Cummins, C., Petoumenos, P., Murray, A., & Leather, H. (2018, July). Compiler fuzzing through deep learning. In Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis (pp. 95-105).
- Farhana, E., Imtiaz, N., & Rahman, A. Synthesizing Program Execution Time Discrepancies in Julia Used for Scientific Software. In 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME) (pp. 496-500). IEEE.
- Soergel, D. A. (2014). Rampant software errors may undermine scientific results. *F1000Research*, *3*.

Acknowledgements

We would like to thank our CSC-6220 professor, Dr. Akond A. Rahman. This material is based upon work supported by the National Science Foundation Graduate Research Fellowship Program under Grant No. 1649609. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.