Noncommital Commits: Predicting Performance Slowdowns in Version Control History

Daniel Nichols[1], Dilan Gunawardana[1], Aniruddha Marathe[2], Todd Gamblin[2], Abhinav Bhatele[1]
[1]University of Maryland, College Park, [2]Lawrence Livermore National Laboratory

Abstract

Scientific software in high performance computing is becoming increasingly complex both in terms of its size and the number of external dependencies. Correctness and performance issues can become more challenging in actively developed software with increasing complexity. This leads to software developers having to spend larger portions of their time on debugging, optimizing, and maintaining code. Making software optimization and maintenance easier for developers is paramount to accelerating the rate of scientific progress. Fortunately, there is a wealth of data on scientific coding practices available implicitly via version control histories. These contain the state of a code at each stage throughout its development via commit snapshots. Commit snapshots provide dynamic insight into the software development process that static analyses of release tarballs do not. In this poster, we present a methodology for:

- Collecting performance data across version control history
- Pruning Abstract Syntax Trees (AST) based on profiled Calling Context Trees (CCT)
- Computing edit scripts that map one AST to another
- Using deep learning to predict performance degradation based on code changes

Data Collection & Preprocessing

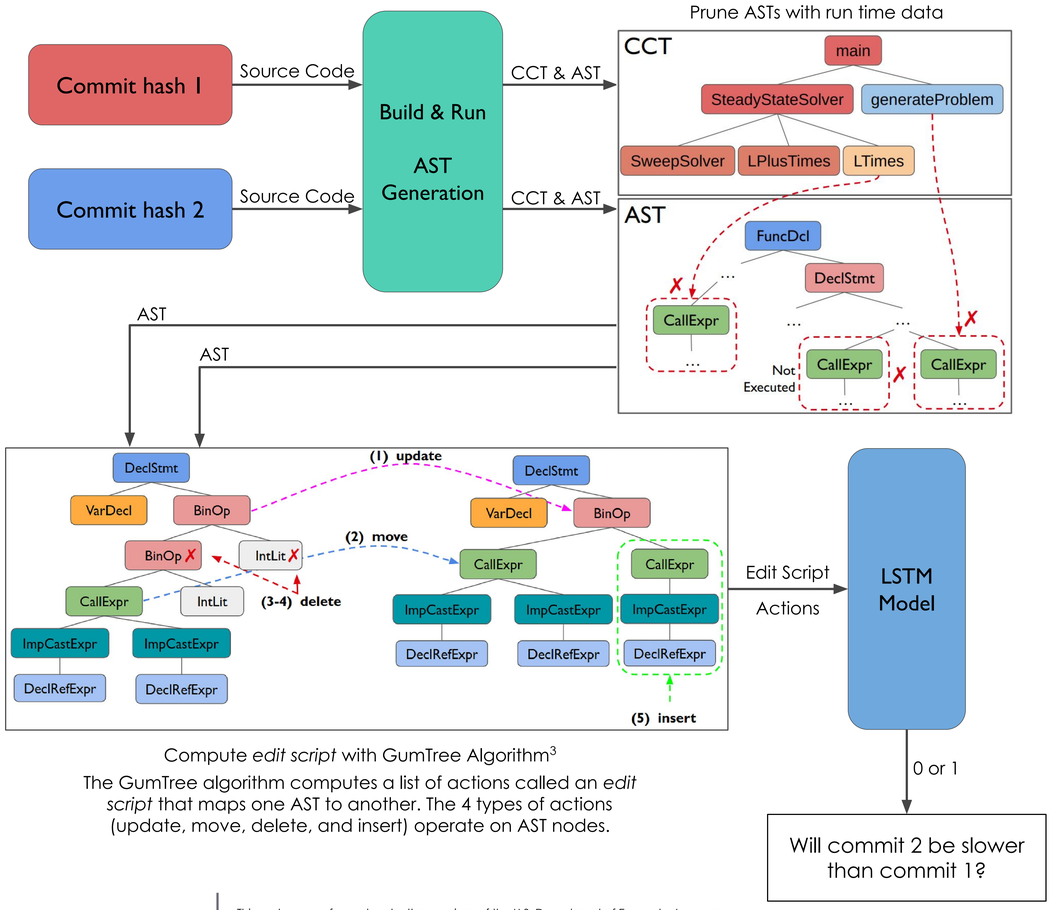
We collect performance data at each commit for 2 proxy applications: Kripke[1] and Laghos[2]. Each commit is run on the Quartz system at Lawrence Livermore National Laboratory (LLNL).

To get more training samples we augment the data set with 3 methods:

- Reflection: swap commits and output label
- Commit Windows: use all commit pairs within a range of commits
- Splitting by Function: split commit by changes per function

Augmentation Method	Kripke Data Set # Samples	Laghos Data Set # Samples
None	107	1168
Reflection	214	2336
Commit Windows	1712	18,688
Splitting by Function	5071	31,140

Machine Learning Pipeline



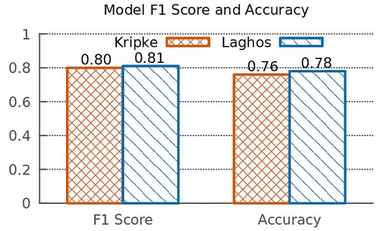
Compute *edit script* with GumTree Algorithm[3]

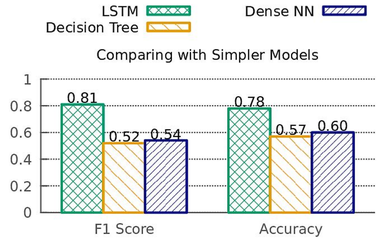
The GumTree algorithm computes a list of actions called an *edit script* that maps one AST to another. The 4 types of actions (update, move, delete, and insert) operate on AST nodes.

Training Results

Long short-term memory (LSTM) model classifies with up to 0.81 F_1 score and 78% accuracy when cross-validated on data set.

The model used is LSTM→Dropout→ Dense with two 256 unit LSTM layers and 0.2 dropout.





Simpler models are trained on summary statistics of edit scripts.

LSTM outperforms simpler models and shows the representative capacity of the *edit scripts*.

Conclusion and Future Work

We have shown that by using data engineering techniques we can utilize deep learning to predict performance degradation in version control history. In future work we plan to:

- Use transfer learning to extend to new applications
- Use causal learning to gain insight into what types of code changes cause performance degradation
- Improve end-result by predicting relative performance
- Suggest code improvements

Acknowledgements

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-POST-838438)

References

[1] https://github.com/LLNL/Kripke
[2] https://github.com/CEED/Laghos
[3] J.-R. Falleri, *et al*, "Fine-grained and accurate source code differencing," in ASE '14.