Harnessing Large Language Models for Simulink Toolchain Testing and Developing Diverse Open-Source Corpora of Simulink Models for Metric and Evolution Analysis

Sohil Lal Shrestha
Computer Science & Eng. Dep.
University of Texas at Arlington
Arlington, Texas, USA

ABSTRACT

MATLAB/Simulink is a de-facto standard tool in several safety-critical industries such as automotive, aerospace, healthcare, and industrial automation for system modeling and analysis, compiling models to code, and deploying code to embedded hardware. On one hand, testing cyber-physical system (CPS) development tools such as MathWorks' Simulink is important as a bug in the toolchain may propagate to the artifacts they produce. On the other hand, it is equally important to understand modeling practices and model evolution to support engineers and scientists as they are widely used in design, simulation, and verification of CPS models. Existing work in this area is limited by two main factors, i.e., (1) inefficiencies of state-of-the-art testing schemes in finding critical tool-chain bugs and (2) the lack of a reusable corpus of public Simulink models. In my thesis, I propose to (1) curate a large reusable corpus of Simulink models to help understand modeling practices and model evolution and (2) leverage such a corpus with deep-learning based language models to test the toolchain.

CCS CONCEPTS

• **Software and its engineering** → **Software testing and debugging**; **Model-driven software engineering**; **Open source model**; • **Computing methodologies** → **Transfer learning**; • **Information systems** → **Language models**; • **Computer systems organization** → *Embedded and cyber-physical systems.*

KEYWORDS

Cyber-physical system development, Simulink, tool chain bugs, deep learning, programming language modeling, GPT-2, mining software repositories, open-source, model evolution

ACM Reference Format:
Sohil Lal Shrestha. 2023. Harnessing Large Language Models for Simulink Toolchain Testing and Developing Diverse Open-Source Corpora of Simulink Models for Metric and Evolution Analysis. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '23), July 17–21, 2023, Seattle, WA, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3597926.3605233

ISSTA '23, July 17–21, 2023, Seattle, WA, USA

ACM ISBN 979-8-4007-0221-1/23/07.
https://doi.org/10.1145/3597926.3605233

1 INTRODUCTION AND MOTIVATION

MATLAB/Simulink is a graphical programming environment for modeling, simulating, and analyzing multi-domain dynamical systems and is a de-facto standard for model-based design. Engineers create Simulink models and compile them to embedded code, often to control safety-critical cyber-physical systems in automotive, aerospace, and healthcare applications. Despite Simulink's importance, there are few large-scale empirical research tackling software engineering problems such as finding bugs in the tool chain and understanding modeling practices and model evolution to better aid development of CPS model using the toolchain.

Recent years have seen an increasing interest in automated validation of tool chains such as Simulink. Although, in software engineering, there are a multitude of ways to find bugs, approaches such as formal verification that guarantees a bug-free toolchain is not applicable to Simulink due to its large code base and lack of complete formal specification, which can be partly attributed to its commercial nature. An alternative is fuzzing [3], which is randomly generating valid or semi-valid inputs that are used to stress test compilers. State-of-the-art Simulink-testing tool SLforge combined randomized fuzzing with differential testing and found 8 new bugs in Simulink [4]. To generate random semi-valid inputs (or Simulink models), SLforge's team parsed semi-formal specifications from Simulink's web page automatically and rigorously incorporated them in SLforge's random model generator.

Although SLforge has demonstrated its effectiveness, it exhibits a dependency on well-documented specifications for updating its random model generator. Furthermore, the engineering effort required to maintain the tool does not scale in response to specification changes or the addition of new features. Furthermore, it is highly desirable to maintain a reasonable level of fidelity with real-world Simulink models when generating models, as the discovery of bugs through the use of such models takes precedence in terms of priority for resolution. Consequently, an alternative approach is desired that effectively overcomes the limitations of SLforge.

In the realm of model-based development research, particularly in the context of MATLAB/Simulink, close collaborations between academia and industry have played a pivotal role. These collaborations often involve the utilization of proprietary models that are not publicly accessible [1]. Moreover, there is a consensus among researchers that publicly available Simulink models developed by third parties are generally limited in size, inadequate in representing real-world models, and difficult to obtain [5, 6, 9–11]. Recent investigations have begun to focus on curating extensive collections of Simulink-based projects sourced from open-source repositories,

enabling valuable insights to be derived from their analysis. For example, Chowdhury et al. [8] curated a corpus of 1,000 Simulink models and conducted a study on the modeling practices and complexity exhibited by open-source Simulink models. Subsequently, Boll et al. [2] independently replicated Chowdhury's study by acquiring the latest version of Chowdhury's Simulink projects, and additionally examined the project's context, size, organization, and evolution to evaluate its suitability for empirical research.

Despite pioneering the use of larger Simulink-based project collections and validating their potential similarity to industrial models, previous attempts at curation were limited by certain factors. These collections were comprised of URLs leading to non-permanent resources, and the models contained therein had ambiguous or unclear licensing information. Additionally, the curation process relied heavily on manual efforts, leading to inconsistencies such as empty projects, duplicate projects, and incomplete metadata. Furthermore, the resulting collections were relatively modest in size, and their curation process may have introduced unintended human errors and biases.

2 MY RELEVANT PUBLICATIONS: DEEPFUZZSL, SLGPT, AND SLNET

Besides contributing to SLEMI, the equivalence-modulo-input (EMI) based Simulink random model generator [6, 7, 16], my main research contributions to date have made significant progress towards creating a large corpus of open-source Simulink models and leveraging a corpus and deep-learning approaches for generating random Simulink models. First, on the deep-learning side, to side-step the age-old problem of missing complete formal specifications of Simulink tool chain and address the limitations of SLforge, that requires significant research and engineering investment, we designed DeepFuzzSL—a novel scheme to infer the Simulink's language validity rules via deep learning based language models called DeepFuzzSL [12, 13]. DeepFuzzSL learns a Long Short Term Memory (LSTM) based language model from existing corpus of Simulink models. Using DeepFuzzSL, we are able to consistently generate over 90% valid Simulink models (on par with the state of the art SLforge) and found 2 bugs, one of which is missed by SLforge.

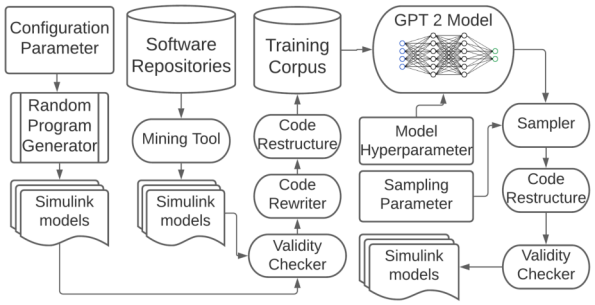

Figure 1: SLGPT obtains Simulink models from a random generator and open-source repositories, simplifies them, and uses them to adapt GPT-2 for finding Simulink crashes.

While deep learning techniques, such as DeepFuzzSL, promise to learn such language specifications from sample models, it needs a large number of training data to work well. As shown in Figure-1, SLGPT [15] addresses this problem by using transfer learning to leverage the powerful Generative Pre-trained Transformer 2 (GPT-2) model, which has been pre-trained on a large set of training data. SLGPT adapts GPT-2 to Simulink with both randomly generated models and models mined from open-source repositories. SLGPT produced Simulink models that are both more similar to open-source models than its closest competitor, DeepFuzzSL, and found a super-set of the Simulink toolchain bugs found by DeepFuzzSL.

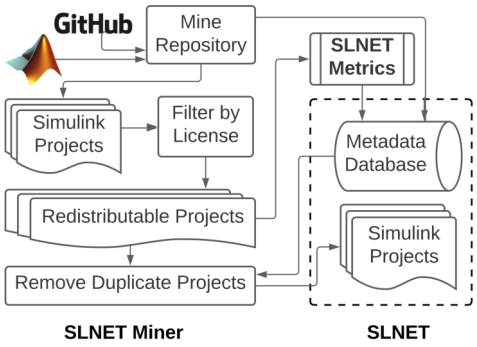

Figure 2: SLNET-Miner collects data, from GitHub and MATLAB Central, removes duplicated projects or those with inappropriate license. SLNET-Metrics extracts model metrics.

Second, on the corpus construction side, existing work using deep learning techniques to generate random Simulink models that test Simulink toolchain is limited by the size and quality of training data. Closely related, deepening our understanding of Simulink models and modeling practices is important as it is crucial to develop tools to better support engineers that use Simulink toolchain for system design. To address the dataset problem, we automate corpus construction and analysis, including data cleaning, metric computation, and packaging–and built SLNET, the largest collection of Simulink models that is fully self-contained and redistributable [14]. SLNET is at least 7 times larger than previous corpora and its open-source data collection tool allows to scale the corpus in the future. Such large corpus of third-party Simulink models may make it easier for engineers and researchers to produce, reproduce, and validate empirical results about Simulink models, modeling practices, and tools that operate on such models.

3 RESEARCH GAP(S) AND QUESTIONS

3.1 Understanding Simulink Modeling Practices

Despite Simulink's importance, the research community's insights on Simulink models and modeling practices remains shallow. The few existing empirical studies of Simulink models have been limited to proprietary models or a small number of public models. Earlier studies on large model corpora suffer from inconsistencies hindering reproduction and generalization of results . To investigate

inconsistencies within earlier studies on Simulink model corpora and modeling practices, we plan to reproduce prior studies, which will help us highlight key issues that future research can learn and benefit from. Then, using a new large-scale corpus SLNET, we will replicate prior studies that used smaller model collections and provide insights on Simulink modeling practices. We focus on the following research questions:

RQ1 Does SLNET have similar metric distributions as the earlier corpora?

RQ2 How do the empirical results obtained in earlier studies on smaller corpora carry over to the larger SLNET corpus?

RQ3 Are SLNET Simulink models configured to be compiled to C/C++ code for deployment on embedded devices?

3.2 Building a Corpus of Open-source Simulink Projects for Evolution Study

Having readily available corpora is crucial for performing replication, reproduction, extension, and verification studies of existing research tools and techniques. SLNET focused on curating a diverse set of Simulink models from different sources and domains but to limit the corpus's size, SLNET only included the latest snapshot of the models or projects. While its metadata links to each project's full revision history, not all projects are still available from the original sources. To support studies relating to model evolution, having a large collection of projects with full revision history is essential. To demonstrate the value of such a corpus, we explore if an evolution study conducted using closed-source project can be done using open-source software. Specifically, we focus on the following research question.

RQ4 Do model changes documented for a single closed-source industrial project generalize to open-source projects?

3.3 Facilitating Simulink Model Search

Having a search engine to find a code snippet or a project has aided software engineering community tremendously. However, existing search engines lack necessary support for meeting the specific demands of engineers searching for Simulink models or projects. To address the issue, we aim to leverage the existing infrastructure of SLNET, that contains automated mining and metric collection tools, to develop a web application that facilitates advanced search capabilities for Simulink models and projects. To evaluate the effectiveness and efficiency of the proposed web application, we focus on the following research questions.

RQ5 What are the specific search requirements and challenges faced by researchers when searching for Simulink models or projects?

RQ6 To what extent does the web application enhance the efficiency of Simulink model/project search in terms of search time and effort?

4 PROPOSED METHODOLOGY

4.1 Understanding Simulink Modeling Practices

Figure-3 presents the application of the ACM's guidelines on reproducibility ("different team, same experimental setup") and replicability ("different team, different experimental setup") to empirical studies conducted on Simulink models. The figure provides an overview of the pertinent variables involved in our study. To ensure the integrity and validity of our research, our methodology consists of two main stages.

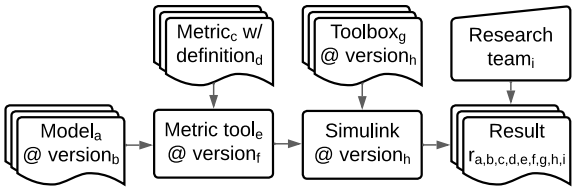

Figure 3: Overview of parameters considered in reproducing and replicating earlier results on Simulink models.

In first stage, we aim to reproduce the results of prior studies by employing the original tools and artifacts. We will establish communication with the original authors to address any uncertainties or inconsistencies, ensuring that our reproduction accurately reflects the findings. The primary motivation behind this reproduction effort is to address concerns regarding unintended human errors and bias, for which empirical evidence has been insufficient.

In the second stage, we will systematically replicate the results obtained from earlier studies using SLNET. To achieve this, we will develop and/or extend the metric collection and analysis tool. This will enable us to obtain repeatable results and facilitate a fair comparison with the findings of previous studies. We will conduct sanity tests on the tools we develop to ensure their consistency with earlier research. Furthermore, we will engage in clarifying discussions with the original researchers to ensure that the tools align with the methodologies employed in the previous studies.

By following this approach, we aim to contribute to the robustness and credibility of empirical research on Simulink models. The systematic reproduction and replication process will help address any inconsistencies, validate prior findings, and provide a foundation for future investigations in this field.

4.2 Building a Corpus of Open-source Simulink Projects for Evolution Study

During the construction of SLNET, our initial approach involved a shallow search on repository hosting sites such as GitHub and included only projects' snapshots to limit the size of SLNET. As a result, SLNET did not support evolution studies since it lacked projects with a revision history. To curate projects suitable for evolution study, we extended SLNET's collection tool to enable the download of the entire Git repository while satisfying GitHub's API limits. We performed an extensive search by downloading repositories (a) that use MATLAB programming language or (b) whose metadata such as the repository name, description, or README file contain the keyword "Simulink," aligning with SLNET's approach. We then filtered the search results by only keeping projects that (1) have Simulink model file that Simulink can open and (2) have a license that allows redistribution.

Our collection process resulted in the acquisition of more than 13,500 projects, which we have named EvoSL$^+$. We systematically

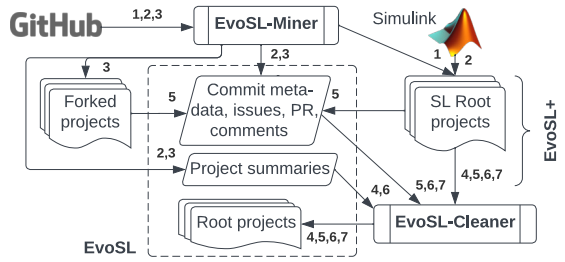

Figure 4: Overview of EvoSL collection and cleaning steps: EvoSL-Miner downloads EvoSL+ (Git projects and metadata), from which EvoSL-Cleaner removes certain Git repositories.

extracted issue tracking metadata and processed the commit metadata of each project to facilitate convenient access and sampling. To ensure data integrity, we removed any duplicate projects and those that exhibited minimal changes in Simulink model files. As a result, we established EvoSL, a curated distribution comprising over 900 projects with 140K commits. This corpus is significant as it is the first extensive collection of open-source models that encompasses both model and project changes and permits redistribution. Leveraging the EvoSL corpus, our research aims to investigate whether observed model evolution changes in a closed-source industrial project are applicable to open-source projects in general.

4.3 Facilitating Simulink Model Search

To gain an understanding of the Simulink model attributes that are of interest to the research community, we will initiate a survey involving researchers and industry experts. By analyzing the survey results, our objective is to identify advanced fine-grained filtering attributes. These attributes will enable users to efficiently sample and locate models that align with their specific requirements.

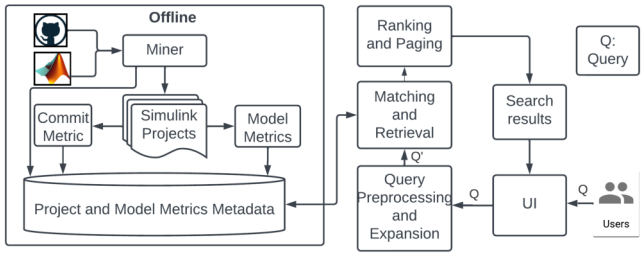

Figure 5: Overview of Simulink model search engine

Figure-5 illustrates the proposed architecture of the web search engine. Leveraging the existing infrastructure of SLNET and EvoSL, we will periodically mine open-source repository hosting sites to gather project, model and commit metrics. These metrics will be stored in a MongoDB database. To enhance the functionality of the tools, we will leverage insights obtained from the survey results to incorporate additional features both in metric collection tools and search user interface.

Users will interact with the web application by utilizing the search interface depicted in Figure 6. Through this interface, they

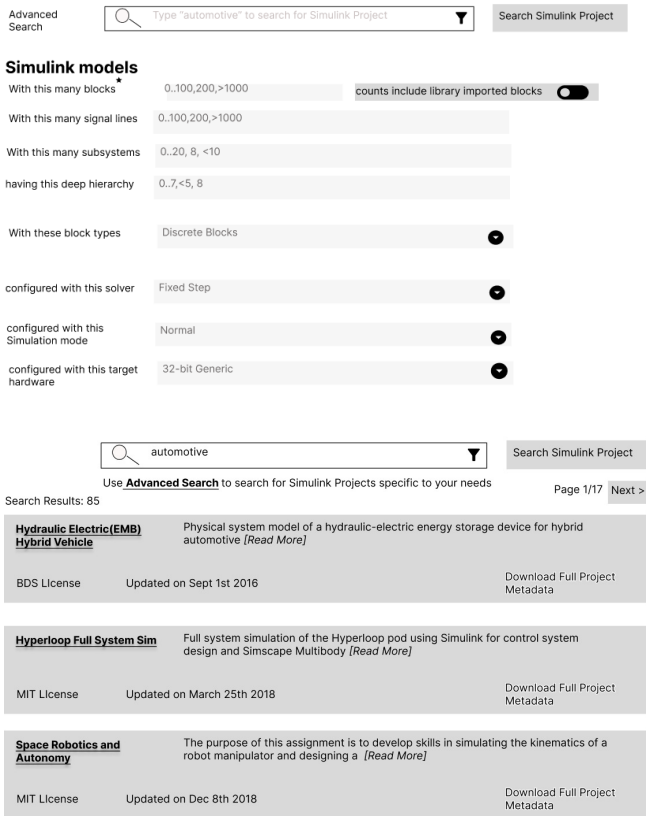

Figure 6: User interface of web app displaying search results

can query the search engine to retrieve relevant information. The development of the web-based search tool will follow an iterative feedback loop, allowing us to continuously incorporate user suggestions, as well as address any reported bugs or issues.

5 CONCLUSIONS

Model based research using Simulink and its scientific progress have been severely limited due to a lack of access to readily available engineered Simulink models. This has lead many research groups to either construct their own evaluation subjects (which raises questions about how these results generalize) or they relied on proprietary models from industry partners (which raises questions about how reproducible such results are). Our work on understanding modeling practices and replicating results will help support the relevance and presence of high quality Simulink models. Our ultimate goal of our corpus collections is to empower the community to perform new and more powerful empirical studies.

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation (NSF) under Grant No. 1911017 and a gift from MathWorks.

REFERENCES

[1] Vincent Bertram, Shahar Maoz, Jan Oliver Ringert, Bernhard Rumpe, and Michael von Wenckstern. 2017. Component and Connector Views in Practice: An Experience Report. In *MODELS*. IEEE Computer Society, 167–177.

[2] Alexander Boll, Florian Brokhausen, Tiago Amorim, Timo Kehrer, and Andreas Vogelsang. 2021. Characteristics, potentials, and limitations of open-source Simulink projects for empirical research. *SoSyM* (2021), 1–20. https://doi.org/10.1007/s10270-021-00883-0

[3] Yang Chen, Alex Groce, Chaoqiang Zhang, Weng-Keen Wong, Xiaoli Z. Fern, Eric Eide, and John Regehr. 2013. Taming compiler fuzzers. In *ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '13, Seattle, WA, USA, June 16-19, 2013*. 197–208. https://doi.org/10.1145/2491956.2462173

[4] Shafiul Azam Chowdhury, Soumik Mohian, Sidharth Mehra, Siddhant Gawsane, Taylor T. Johnson, and Christoph Csallner. 2018. Automatically finding bugs in a commercial cyber-physical system development tool chain with SLforge. In *Proc. 40th ACM/IEEE International Conference on Software Engineering (ICSE)*. ACM.

[5] Shafiul Azam Chowdhury, Soumik Mohian, Sidharth Mehra, Siddhant Gawsane, Taylor T. Johnson, and Christoph Csallner. 2018. Automatically finding bugs in a commercial cyber-physical system development tool chain with SLforge. In *ICSE*. ACM, 981–992. https://doi.org/10.1145/3180155.3180231

[6] Shafiul Azam Chowdhury, Sohil Lal Shrestha, Taylor T. Johnson, and Christoph Csallner. 2020. SLEMI: Equivalence modulo input (EMI) based mutation of CPS models for finding compiler bugs in Simulink. In *ICSE*. ACM, 335–346. https://doi.org/10.1145/3377811.3380381

[7] Shafiul Azam Chowdhury, Sohil Lal Shrestha, Taylor T. Johnson, and Christoph Csallner. 2020. SLEMI: Finding Simulink Compiler Bugs through Equivalence Modulo Input (EMI). In *Proc. 42nd International Conference on Software Engineering (ICSE), Companion Volume*. ACM, 1–4. https://doi.org/10.1145/3377812.3382147

[8] Shafiul Azam Chowdhury, Lina Sera Varghese, Soumik Mohian, Taylor T. Johnson, and Christoph Csallner. 2018. A curated corpus of Simulink models for model-based empirical studies. In *SEsCPS*. ACM, 45–48. https://doi.org/10.1145/3196478.3196484

[9] Zhenying Jiang, Xiao Wu, Zeqian Dong, and Ming Mu. 2017. Optimal Test Case Generation for Simulink Models Using Slicing. In *QRS-C*. 363–369. https://doi.org/10.1109/QRS-C.2017.67

[10] Reza Matinnejad, Shiva Nejati, Lionel C. Briand, and Thomas Bruckmann. 2015. Effective test suites for mixed discrete-continuous stateflow controllers. In *ESEC/FSE*. ACM, 84–95. https://doi.org/10.1145/2786805.2786818

[11] A. Chakrapani Rao, A. Raouf, Gunwant Dhadyalla, and V. Pasupuleti. 2017. Mutation Testing Based Evaluation of Formal Verification Tools. In *DSA*. IEEE, 1–7. https://doi.org/10.1109/DSA.2017.10

[12] Sohil Lal Shrestha. 2020. Automatic generation of Simulink models to find bugs in a cyber-physical system tool chain using deep learning. In *Proc. 42nd International Conference on Software Engineering (ICSE), Companion Volume*. ACM, 110–112. https://doi.org/10.1145/3377812.3382163

[13] Sohil Lal Shrestha, Shafiul Azam Chowdhury, and Christoph Csallner. 2020. DeepFuzzSL: Generating models with deep learning to find bugs in the Simulink toolchain. In *DeepTest 2020*. ACM.

[14] Sohil Lal Shrestha, Shafiul Azam Chowdhury, and Christoph Csallner. 2022. SLNET: A Redistributable Corpus of 3rd-party Simulink Models. In *19th IEEE/ACM International Conference on Mining Software Repositories, MSR 2022, Pittsburgh, PA, USA, May 23-24, 2022*. ACM, 1–5. https://doi.org/10.1145/3524842.3528001

[15] Sohil Lal Shrestha and Christoph Csallner. 2021. SLGPT: Using Transfer Learning to Directly Generate Simulink Model Files and Find Bugs in the Simulink Toolchain. In *EASE 2021: Evaluation and Assessment in Software Engineering, Trondheim, Norway, June 21-24, 2021*. ACM, 260–265. https://doi.org/10.1145/3463274.3463806

[16] Sohil Lal Shrestha, Saroj Panda, and Christoph Csallner. 2018. Complementing Machine Learning Classifiers via Dynamic Symbolic Execution: "Human vs. Bot Generated" Tweets. In *6th IEEE/ACM International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering, RAISE@ICSE 2018, Gothenburg, Sweden, May 27, 2018*, Walter F. Tichy and Leandro L. Minku (Eds.). ACM, 15–20. https://doi.org/10.1145/3194104.3194111

Received 2023-05-24; accepted 2023-06-07