SLNET: A Redistributable Corpus of 3rd-party Simulink Models

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

Shafiul Azam Chowdhury
Computer Science & Eng. Dep.
University of Texas at Arlington
Arlington, Texas, USA

Christoph Csallner
Computer Science & Eng. Dep.
University of Texas at Arlington
Arlington, Texas, USA

ABSTRACT

MATLAB/Simulink is widely used 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 Simulink studies, perhaps because there is no large readily available corpus of third-party open-source Simulink models. To enable empirical Simulink studies, this paper introduces SLNET, the largest corpus of freely available third-party Simulink models. SLNET has several advantages over earlier collections. Specifically, SLNET is 8 times larger than the largest previous corpus of Simulink models, includes fine-grained metadata, is constructed automatically, is self-contained, and allows redistribution. SLNET is available under permissive open-source licenses and contains its collection and analysis tools.

CCS CONCEPTS

• **Software and its engineering** → **Software libraries and repositories**; **Model-driven software engineering**; • **Computer systems organization** → *Embedded and cyber-physical systems.*

KEYWORDS

Simulink, mining software repositories, open-source

ACM Reference Format:
Sohil Lal Shrestha, Shafiul Azam Chowdhury, and Christoph Csallner. 2022. SLNET: A Redistributable Corpus of 3rd-party Simulink Models. In *19th International Conference on Mining Software Repositories (MSR '22), May 23–24, 2022, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3524842.3528001

1 INTRODUCTION

Currently there is no collection of Simulink models that is commonly used in empirical studies. Though there have been previous model collections, they lack fine-grained meta-information, are not self-contained, and are not redistributable due to restrictive or missing licenses—making them hard or impossible to use for most empirical researchers. Given the lack of such a collection, the few existing empirical studies of Simulink models have been limited to proprietary models or a small number of public models [9, 41, 42].

Deepening our understanding of Simulink models and modeling practices is important, as 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 [44, 46]. Having a 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.

The most closely related previous work has studied an initial collection of 391 third-party Simulink models [9] and later extended it to a curated corpus ("SLC0") of some 1k third-party Simulink models [11]. Boll et al. [4] collected an updated version of SLC0 and assessed the corpus's suitability for empirical research. While pioneering larger studies and validating that models from such a corpus can be similar to industrial models, these collections consisted of a list of URLs to non-permanent resources [9] and contained models with unclear license information [11]. These collections were largely manual, which lead to inconsistencies (empty projects, duplicate projects, and missing metadata), relatively modest collection size, and may yield unintended human errors and bias.

To address these limitations, SLNET automates corpus construction and analysis, including data acquisition, cleaning (except for the rarely required manual review of a new license type), metric computation, and packaging. SLNET thereby automatically mines and analyses Simulink models from the two most popular repositories for sharing Simulink models, yielding a collection of thousands of models that is fully self-contained and allows redistribution.

To allow fine-grained selection of Simulink models and projects, SLNET computes several project-level and model-level metrics [4] and exposes them in a SQL database. SLNET similarly identifies and labels libraries and models that are test harnesses [31]. To summarize, this paper makes the following major contributions.

- SLNET is redistributable and 8 times larger than the prior largest known corpus of third-party Simulink models.
- SLNET [38] and its tools [36, 37] are available under permissive open-source licenses (CC BY and BSD 3-clause), e.g., SLNET is at: **https://doi.org/10.5281/zenodo.5259648**

2 BACKGROUND ON SIMULINK

Simulink [23] is a widely used commercial tool-chain for model-based design [44, 46][1]. Engineers typically design a cyber-physical system (CPS) model in Simulink's graphical modeling environment. A Simulink model such as Figure 1 is a *block diagram*, where each block represents equations or modeling components. Depending on the block type, each block can accept input (via input ports), perform some operation on its inputs, and produce output (via output ports), which then can optionally be forwarded to other blocks via explicit or implicit connection lines (aka signal lines).

MSR '22, May 23–24, 2022, Pittsburgh, PA, USA

ACM ISBN 978-1-4503-9303-4/22/05.
https://doi.org/10.1145/3524842.3528001

[1] Searching for "Simulink" jobs on LinkedIn in the US currently yields over 5k job postings: https://www.linkedin.com/jobs/search/?keywords="simulink"&location=US

Simulink users can add blocks from various built-in *libraries* and toolboxes [25], and can also define *custom blocks* in "native" code (e.g., in C) using the S-function interface.

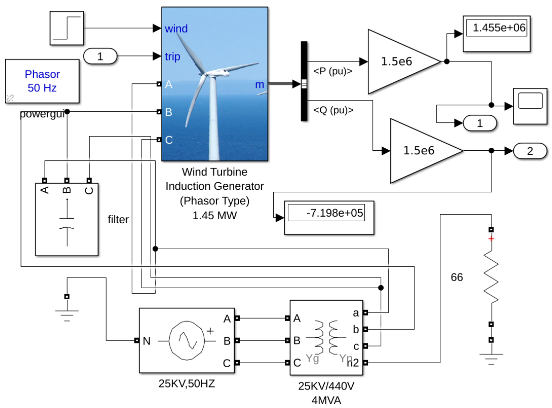

Figure 1: Sample SLNET Simulink model of a 1.5MW wind generation plant [30] with 18 blocks and 23 connections.

To deal with model size, users can create hierarchical models, by (recursively) grouping blocks in (a) a *Subsystem* or (b) in a separate model via *Model Reference*. Simulink does not permit a cyclic model hierarchy, but there may be block connection (aka data dependence) cycles, including algebraic loops[2].

As a first step, *compiling* translates the model into a toolchain-internal representation. When *simulating* the compiled model, the toolchain computes the output of each block at successive time steps over a specified time range using pre-configured numerical solvers. *Fixed-step* solvers solve the model at fixed time intervals whereas *variable-step* solvers automatically adjust the time intervals at which the model is solved. Simulink may reject a model if it cannot numerically solve an algebraic loop. Simulink offers different simulation modes, i.e., *Normal* mode "only" simulates blocks, *Accelerator* speeds up simulation by emitting native code, and *Rapid Accelerator* produces a standalone executable[3].

3 SLNET DESIGN & CONSTRUCTION

SLNET is not a superset of earlier Simulink corpora [4, 11] as earlier corpora were neither self-contained nor redistributable. Figure 2 gives an overview of SLNET's construction. We built SLNET from models shared in GitHub [16] and MATLAB Central [24]. Due to time limitations we do not collect Simulink models from smaller repositories such as GitLab [17] and SourceForge [40]. Before removing projects that are empty, duplicate, or have an unclear license, a quick search for "Simulink" yields some 60 GitLab and some 70 SourceForge projects.

While GitHub offers commit-level version control, MATLAB Central "only" serves project releases. To limit SLNET's size and due to the different versioning (git commits vs. project releases), in February 2020 we "only" collected Simulink project snapshots (i.e., all current project files plus project metadata).

[2]https://www.mathworks.com/help/simulink/ug/algebraic-loops.html
[3]Simulink's embedded code generation workflow for deployment on target platforms is distinct from these simulation modes.

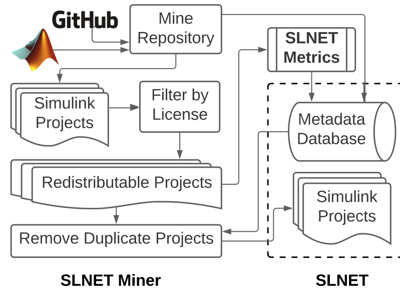

Figure 2: Overview: SLNET-Miner collects files and data, removes empty and duplicated projects or those without appropriate license. SLNET-Metrics extracts model metrics.

GitHub provides a REST API to discover projects and extract them with their metadata. SLNET-Miner queries the GitHub API (via PyGithub [43]) with the keyword "Simulink". Unlike previous work [9, 11], we used keyword search and not file extension search, as file extension search is typically intended to search within a given GitHub repository and using file extension search in GitHub's search page produced many false positives.

The GitHub API expose 23 types of project-level information [15], of which SLNET retains 20. The other 3 are redundant (full project name) or API-internal (API query relevance score and node id). From the API we also obtain each project's topics (user-created labels and tags). From the downloaded project files, we extracted the list of Simulink model files plus the project's license.

As MATLAB Central "only" offers an RSS feed [22] for its file exchange platform, we filter the search result feed by Simulink models and then parse the feed to collect each project's download URL plus 14 other types of project metadata. Since from the RSS feed we could not construct the download URL for all projects, we extracted 2,941 of the 3,110 available projects.

3.1 Data Cleaning & Storage: ZIP + SQLite

We remove projects without Simulink models (i.e., file extensions *slx* or *mdl*) and projects we know to contain synthetic models (i.e., model generators [9, 10]). We heuristically search for other model generators (via terms "automat", "random", "fuzz", and "generate") in project titles, project descriptions, and project tags, which yielded 530 projects (e.g., on fuzzy logic). As we did not find evidence that these projects generate models we kept them in SLNET.

Table 1: Data cleaning: Real = has 1+ models (likely non-synthetic); License = has a license; SLNET+D = license allows redistribution; SLNET = has a model with 1+ blocks after removing potential duplicate projects; Model counts here include 1,130 library and 9 test harness models.

	Projects				Models
	Real	License	SLNET+D	SLNET	SLNET
GitHub	1,284	232	231	225	2,088
MATLAB Cl	2,941	2,746	2,728	2,612	7,029
Total	4,225	2,978	2,959	2,837	9,117

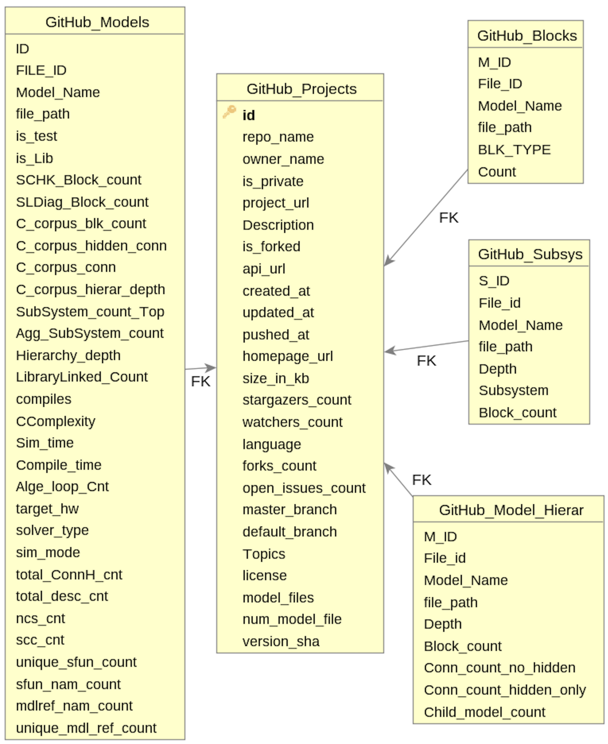

Figure 3: SLNET database schema (GitHub portion). The MATLAB Central portion only differs in its _Projects table [38].

We then remove projects without a license or whose license does not allow redistribution. GitHub has a structured way for authors to set a license, which GitHub converts to a file (and exposes via an API). We manually reviewed the remaining 50 projects' licenses (where GitHub did not understand the author's license or for MATLAB Central projects without a BSD license).

We heuristically remove potentially duplicate projects. We consider project A a duplicate of B if (1) A and B contain the same number of Simulink model files and (2) there is a bijective mapping between models in A and B based on our Section 3.2 model metrics (excluding compile time). If A and B are from the same data source (GitHub or MATLAB Central), we keep the first-created one in SLNET. Otherwise, we keep the one from GitHub, as it offers more fine-grained meta-data. Finally, we remove dummy projects (projects whose Simulink models all have zero blocks).

Table 1 summarizes data cleaning. After removing model generators we downloaded 4,225 projects with at least one Simulink model, of which 2,978 had a license, of which 2,959 allowed redistribution. Removing 112 potentially duplicate plus 10 dummy projects yielded 2,837 projects and their 9,117 Simulink models in SLNET.

SLNET is on Zenodo (a second archive contains the 112 duplicate projects) [38]. Each project has a snapshot of its files in a ZIP archive in either the GitHub or MATLAB Central directory. Each project is named **ID.zip**, where ID is an identifier defined by GitHub or MATLAB Central. SLNET includes the Figure 3 SQLite[4] database. It contains project-level information (license type, etc.) from the source repositories and the model metrics our tools extracted. Users can thus select models and projects from SLNET via SQL queries.

3.2 Project & Model Metrics

Table 2: SLNET's project engagement distributions are long-tailed as in other studies of open-source projects [2, 18–20].

	Metadata	Min	Max	Avg	Med.	SD
	Stargazers	0	128	3.5	0	12.1
GitHub	Forks	0	122	2.8	0	10.7
	Open Issues	0	82	1.2	0	6.5
	Comments	0	218	3.5	1	12.3
MATLAB Cl	Ratings	0	108	2.9	1	6.8
	Avg. Rating	0	5	2.5	3	2.2

To get an insight into the projects' domain and popularity we first searched the user-generated project tags (i.e., GitHub "topics" and MATLAB Central "categories") for common domains (i.e., the Simulink project domains identified by Boll et al.[4]), yielding Electronics (983), Automotive (64), Communications (61), Robotics (52), Energy (48), Aerospace (47), Biotech (20), and Medicine (2). Table 2 shows data often used as proxies for project popularity or engagement (e.g., people who have star-ed or forked a GitHub project or provided a 1–5 star rating for a MATLAB Central project). For example, a SLNET GitHub project has on average 2.8 forks.

To extract commonly used model metrics (such as number of blocks, connections, subsystems, and linked blocks[5]) we implemented the SLNET-Metrics tool [36] on top of Simulink's APIs. While our Simulink installation and toolbox configuration [35] cannot compile a significant portion of SLNET models (mostly due to missing toolbox licenses), these APIs still compute metrics for these non-compiling models, except for three metrics (algebraic loops, cyclomatic complexity, and compile time).

SLNET-Metrics failed to compute metrics for 88/9,117 models (21 from GitHub, 67 from MATLAB Central). Most of these 88 were due to Simulink version issues (missing Simulink toolboxes, model name conflicts with a keyword or toolbox file name) and bugs introduced by manually-edited model files. SLNET does not include metrics for these 88 models and thus also ignores them for the above duplicate-via-bijection removal.

SLNET-Metrics collects each model's hierarchical depth, solver type, simulation mode, target hardware, and use of S-functions and model references. While SLNET models contain elements from the state-machine toolbox Stateflow, Stateflow is out of scope and our metrics do not count the Stateflow-contents of a Simulink block.

Unlike SLC0, SLNET-Metrics does not count nested blocks imported from libraries or their connections (aka "masked subsystems"). This mirrors procedural code metrics, which also do not count LOC a program imports from a library. As SLC0's counting of such imported blocks approximates the model's overall conceptual

[4]SQLite is widely used, free, self-contained, server-less, zero-configuration, backwards compatible, and cross-platform: https://www.sqlite.org/index.html

[5]https://www.mathworks.com/help/simulink/ug/creating-and-working-with-linked-blocks.html

	Models		Hierarchical		Blocks		Connections		Solver Step		Simulation Mode			
Source	M	Mc	Mh	Mh^{t0}	B	B^{t0}	C	C^{t0}	Fixed	Var	Nor	Ext	PIL	Ac
GitHub	1,639	541	878	1,304	190,321	414,241	188,285	395,725	860	762	1,501	103	2	14
MATLAB Cl	6,251	3,636	3,893	5,566	838,956	3,197,221	915,975	3,084,605	1,757	4,493	5,984	186	2	76
Total	7,890	4,177	4,771	6,870	1,029,277	3,611,462	1,104,260	3,480,330	2,617	5,255	7,485	289	4	90

Table 3: SLNET's model metrics after removing library & test harness models; M = models; Mc = models we could readily compile; Mh = hierarchical models (readily compilable and otherwise); C = non-hidden connections; ^{t0} = via SLC0's metric tool; Var = variable; Nor = normal; Ext = external; PIL = processor in the loop; Ac = accelerator. For 18 models the API did not indicate simulation mode or solver type. The remaining 4 models are configured for Rapid Accelerator simulation mode.

complexity [33], Table 3 also includes these counts. As an example, the Figure 1 model imports blocks from the Simscape toolbox, yielding a SLC0-style block count of 907 with 919 connections.

The Simulink API labels only 9 SLNET models as a test harness, likely because many open-source projects do not have the required "Simulink Test" license to develop such tests. Beyond this official classification SLNET contains likely "work-around" test harnesses. The SLC0 metrics tool heuristically matches model and folder names with "test" and "harness" and SLNET labels such models separately.

We performed sanity checks on the model metrics other papers reported about industrial models (block count, etc.). We also randomly sampled from the top 100 largest models in SLNET. Based on the sampled models' documentation we are confident that these were real human-created (non-synthetic) models.

4 POTENTIAL RESEARCH DIRECTIONS

Since most industrial models are proprietary SLNET is unlikely to reflect their distribution. Instead, the goal is to provide the largest possible redistributable self-contained corpus of non-synthetic models. Different research projects will require different SLNET subsets (e.g., many small models for training deep-learning classifiers vs. large models to evaluate a technique's scalability), which the SQL metadata database facilitates. Having more models is better, especially in deep learning, but also when trying to understand the breadth of modelling practices, or when looking for edge cases (e.g., to test model analysis tools). Following are example directions.

While there has been significant interest in other software engineering areas [6, 21, 45], applying machine learning is relatively under-explored in model driven engineering [1, 8]. To work well, many machine learning and deep learning algorithms require large training sets. SLNET with its many models and rich metadata is thus well-suited. For example, a SLNET subset has been used to train a deep learning model for random Simulink model generation, to find bugs in the Simulink toolchain [39]. Due to their smaller size, this would have not been possible with the earlier corpora.

Due to the lack of easily available open-source models that fit certain characteristics, recent work reverted to evaluating tools on synthetic models [10]. SLNET offers a complimentary (and often preferred) evaluation option with human-authored models.

Recent work including in clone detection, refactoring, model slicing, and model smells has relied on evaluations with few proprietary Simulink models [5, 13, 14, 29, 32]. For example, Deissenboeck et al. [13] evaluated their clone detection approach on a single proprietary Simulink model with 20k blocks. Complementing such evaluations with a variety of open-source models from SLNET could make such studies more general and easier to replicate.

Understanding modeling practices would enable researchers to tune their tools to how engineers use Simulink in various settings. For example, SLforge guides its random model generation by how often blocks appear in 391 open-source models [9]. The larger size of SLNET could thus, e.g., yield useful insights for tool design.

There may also be interesting correlations between metrics, maybe connecting model metrics to project metrics (e.g., model size metrics with project engagement). More generally, SLNET could contribute to a deeper understanding of model modularity, comprehension, quality, and maintainability [3, 12, 28, 34].

While SLNET is unlikely to exactly represent closed-source development, the precise shape of this relation is an open question. For example, for the related domain of Object Constraint Language (OCL) expressions [7], Mengerink et al. found the distribution of expression complexity mined from GitHub projects reflects the distribution in closed-source projects, so open-source projects can be used as a proxy for industrial projects [26, 27].

5 THREATS TO VALIDITY

Due to its search heuristics SLNET-Miner may miss Simulink models (e.g., by missing some of the non-documented RSS feed URLs). Furthermore, since SLNET contains only redistributable projects, results may not be representative of all open source Simulink projects. On the flip side, while removing forks and duplicates, SLNET likely contains clones (from near-duplicate projects to adapted model portions), which can be an opportunity for clone-based research (and a challenge for others). Finally, SLNET-Metrics calls the Check API of Simulink R2019b. While this API has been available since Simulink R2017b, its behavior may change across releases and thus yield different metric values in future Simulink versions.

6 CONCLUSIONS

SLNET is the largest corpus of freely available third-party Simulink models. SLNET is 8 times larger than the largest previous Simulink corpus, includes fine-grained metadata, is constructed automatically, is self-contained, and allows redistribution.

ACKNOWLEDGMENTS

Christoph Csallner has a potential research conflict of interest due to a financial interest with Microsoft and The Trade Desk. A management plan has been created to preserve objectivity in research in accordance with UTA policy. This material is based upon work supported by the National Science Foundation (NSF) under Grant No. 1911017 and a gift from MathWorks.

REFERENCES

[1] Bhisma Adhikari. 2021. Intelligent Simulink Modeling Assistance Via Model Clones and Machine learning. , 237 pages.

[2] Mohammad Y. Allaho and Wang-Chien Lee. 2014. Trends and behavior of developers in open collaborative software projects. In *Proc. International Conference on Behavioral, Economic, and Socio-Cultural Computing (BESC)*. IEEE, 96–102. https://doi.org/10.1109/BESC.2014.7059515

[3] Erik Aceiro Antonio, Fabiano Ferrari, Glauco A de P Caurin, and Sandra CPF Fabbri. 2014. A set of metrics for characterizing simulink model comprehension. *Journal of Computer Science and Technology* 14, 02 (2014), 88–94.

[4] Alexander Boll, Florian Brokhausen, Tiago Amorim, Timo Kehrer, and Andreas Vogelsang. 2021. Characteristics, potentials, and limitations of open-source Simulink projects for empirical research. *Software and Systems Modeling* (2021), 1–20.

[5] Alexander Boll and Timo Kehrer. 2020. On the Replicability of Experimental Tool Evaluations in Model-Based Development - Lessons Learnt from a Systematic Literature Review Focusing on MATLAB/Simulink. In *Proc. 1st International Conference on Systems Modelling and Management (ICSMM)*. Springer, 111–130. https://doi.org/10.1007/978-3-030-58167-1_9

[6] Marcel Bruch, Martin Monperrus, and Mira Mezini. 2009. Learning from examples to improve code completion systems. In *Proc. ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE)*. ACM, 213–222. https://doi.org/10.1145/1595696.1595728

[7] Jordi Cabot and Martin Gogolla. 2012. Object Constraint Language (OCL): A Definitive Guide. In *Proc. 12th International School on Formal Methods for the Design of Computer, Communication, and Software Systems*. Springer, 58–90. https://doi.org/10.1007/978-3-642-30982-3_3

[8] Yuqi Chen, Christopher M. Poskitt, Jun Sun, Sridhar Adepu, and Fan Zhang. 2019. Learning-Guided Network Fuzzing for Testing Cyber-Physical System Defences. In *34th IEEE/ACM International Conference on Automated Software Engineering, ASE 2019, San Diego, CA, USA, November 11-15, 2019*. IEEE, 962–973. https://doi.org/10.1109/ASE.2019.00093

[9] 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, 981–992.

[10] Shafiul Azam Chowdhury, Sohil L 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 *Proc. 42nd ACM/IEEE International Conference on Software Engineering (ICSE)*. ACM, 335–346.

[11] 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 *Proc. 4th International Workshop on Software Engineering for Smart Cyber-Physical Systems (SEsCPS)*. ACM, 45–48.

[12] Yanja Dajsuren, Mark G. J. van den Brand, Alexander Serebrenik, and Serguei A. Roubtsov. 2013. Simulink models are also software: modularity assessment. In *Proc. 9th International ACM SIGSOFT conference on Quality of Software Architectures (QoSA)*. 99–106. https://doi.org/10.1145/2465478.2465482

[13] Florian Deissenboeck, Benjamin Hummel, Elmar Jürgens, Bernhard Schätz, Stefan Wagner, Jean-Francois Girard, and Stefan Teuchert. 2008. Clone detection in automotive model-based development. In *30th International Conference on Software Engineering (ICSE 2008), Leipzig, Germany, May 10-18, 2008*. 603–612. https://doi.org/10.1145/1368088.1368172

[14] Thomas Gerlitz, Quang Minh Tran, and Christian Dziobek. 2015. Detection and Handling of Model Smells for MATLAB/Simulink models. In *Proc. International Workshop on Modelling in Automotive Software Engineering*, Vol. 1487. CEUR-WS.org, 13–22.

[15] GitHub Inc. 2020. GitHub Developer. https://developer.github.com/v3/search/ February 2020.

[16] GitHub Inc. 2021. GitHub. https://github.com Accessed Jan 2022.

[17] GitLab. 2021. GitLab. https://gitlab.com Accessed May 2021.

[18] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M. Germán, and Daniela E. Damian. 2016. An in-depth study of the promises and perils of mining GitHub. *Empirical Software Engineering* 21, 5 (2016), 2035–2071. https://doi.org/10.1007/s10664-015-9393-5

[19] Antonio Lima, Luca Rossi, and Mirco Musolesi. 2014. Coding Together at Scale: GitHub as a Collaborative Social Network. In *Proc. 8th International Conference on Weblogs and Social Media (ICWSM)*. AAAI.

[20] Wanwangying Ma, Lin Chen, Yuming Zhou, and Baowen Xu. 2016. What Are the Dominant Projects in the GitHub Python Ecosystem?. In *Proc. 3rd International Conference on Trustworthy Systems and their Applications (TSA)*. IEEE, 87–95. https://doi.org/10.1109/TSA.2016.23

[21] Ruchika Malhotra. 2015. A systematic review of machine learning techniques for software fault prediction. *Appl. Soft Comput.* 27 (2015), 504–518. https://doi.org/10.1016/j.asoc.2014.11.023

[22] MathWorks Inc. 2021. MathWorks RSS. https://www.mathworks.com/company/rss.html February 2020.

[23] MathWorks Inc. 2021. MATLAB & Simulink. https://www.mathworks.com/products/simulink.html/ Accessed Jan 2022.

[24] MathWorks Inc. 2021. MATLAB Central. https://www.mathworks.com/matlabcentral/fileexchange/ Accessed Jan 2022.

[25] MathWorks Inc. 2021. Simulink Block Libraries Documentation. https://www.mathworks.com/help/simulink/block-libraries.html Accessed Jan 2022.

[26] Josh G. M. Mengerink, Jeroen Noten, Ramon R. H. Schiffelers, Mark G. J. van den Brand, and Alexander Serebrenik. 2017. A Case of Industrial vs. Open-source OCL: Not So Different After All. In *Proc. MODELS Posters*, Vol. 2019. CEUR-WS.org, 472–474.

[27] Jeroen Noten, Josh Mengerink, and Alexander Serebrenik. 2017. A data set of OCL expressions on GitHub. In *Proc. 14th International Conference on Mining Software Repositories (MSR)*. IEEE, 531–534. https://doi.org/10.1109/MSR.2017.52

[28] Marta Olszewska, Yanja Dajsuren, Harald Altinger, Alexander Serebrenik, Marina A. Waldén, and Mark G. J. van den Brand. 2016. Tailoring complexity metrics for Simulink models. In *Proc. 10th European Conference on Software Architecture Workshops*. 5.

[29] Vera Pantelic, Steven M. Postma, Mark Lawford, Monika Jaskolka, Bennett Mackenzie, Alexandre Korobkine, Marc Bender, Jeff Ong, Gordon Marks, and Alan Wassyng. 2018. Software engineering practices and Simulink: bridging the gap. *STTT* 20, 1 (2018), 95–117. https://doi.org/10.1007/s10009-017-0450-9

[30] Gaddala Jaya Raju. 2019. 1.5MW Wind Generation Plant. https://www.mathworks.com/matlabcentral/fileexchange/73469-1-5mw-wind-generation-plant. Accessed Jan 2022.

[31] Eric J. Rapos and James R. Cordy. 2018. SimEvo: A Toolset for Simulink Test Evolution & Maintenance. In *11th IEEE International Conference on Software Testing, Verification and Validation, ICST 2018, Västerås, Sweden, April 9-13, 2018*. IEEE Computer Society, 410–415. https://doi.org/10.1109/ICST.2018.00049

[32] Robert Reicherdt and Sabine Glesner. 2012. Slicing MATLAB Simulink models. In *34th International Conference on Software Engineering, ICSE 2012, June 2-9, 2012, Zurich, Switzerland*. 551–561. https://doi.org/10.1109/ICSE.2012.6227161

[33] Guy Rouleau. 2021. How many blocks are in that model? https://blogs.mathworks.com/simulink/2009/08/11/how-many-blocks-are-in-that-model Accessed Jan 2022.

[34] Jan Schroeder, Christian Berger, Thomas Herpel, and Miroslaw Staron. 2015. Comparing the Applicability of Complexity Measurements for Simulink Models during Integration Testing - An Industrial Case Study. In *Proc. 2nd IEEE/ACM International Workshop on Software Architecture and Metrics (SAM)*. IEEE, 35–40. https://doi.org/10.1109/SAM.2015.12

[35] Sohil L Shrestha. 2021. MATLAB Simulink version Info. https://github.com/50417/SLNET_Metrics/wiki/MATLAB-Simulink-Installation Accessed Jan 25 2022.

[36] Sohil L Shrestha. 2022. *50417/SLNET_Metrics: SLNET_Metrics MSR Release*. https://doi.org/10.5281/zenodo.6336048

[37] Sohil L Shrestha. 2022. *50417/SLNet_Miner: SLNET-Miner MSR Release*. https://doi.org/10.5281/zenodo.6336034

[38] Sohil Lal Shrestha, Shafiul Azam Chowdhury, and Christoph Csallner. 2022. *SLNET: A Redistributable Corpus of 3rd-party Simulink Models*. https://doi.org/10.5281/zenodo.5259648

[39] 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.

[40] Slashdot Media. 2021. SourceForge. https://sourceforge.net/directory/?q=simulink Accessed May 2021.

[41] Matthew Stephan, Manar H. Alalfi, James R. Cordy, and Andrew Stevenson. 2013. Evolution of Model Clones in Simulink. In *Proc. Workshop on Models and Evolution*. 40–49.

[42] Matthew Stephan and James R. Cordy. 2015. Identification of Simulink model antipattern instances using model clone detection. In *18th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MoDELS 2015*. 276–285. https://doi.org/10.1109/MODELS.2015.7338258

[43] Vincent Jacques. 2007. PyGitHub. https://pygithub.readthedocs.io/en/latest/introduction.html Accessed Jan 2022.

[44] Marilyn Wolf and Eric Feron. 2015. What don't we know about CPS architectures?. In *Proc. 52nd Annual Design Automation Conference (DAC)*. 80:1–80:4. https://doi.org/10.1145/2744769.2747950

[45] Marco Zanoni, Francesca Arcelli Fontana, and Fabio Stella. 2015. On applying machine learning techniques for design pattern detection. *J. Syst. Softw.* 103 (2015), 102–117. https://doi.org/10.1016/j.jss.2015.01.037

[46] Xi Zheng, Christine Julien, Miryung Kim, and Sarfraz Khurshid. 2017. Perceptions on the State of the Art in Verification and Validation in Cyber-Physical Systems. *IEEE Systems Journal* 11, 4 (2017), 2614–2627. https://doi.org/10.1109/JSYST.2015.2496293