Replicability Study: Corpora For Understanding Simulink Models & Projects

Sohil Lal Shrestha
Computer Science & Eng. Dept.
University of Texas at Arlington
Arlington, TX 76019, USA
sohil.shrestha@mavs.uta.edu

Shafiul Azam Chowdhury
Computer Science & Eng. Dept.
University of Texas at Arlington
Arlington, TX 76019, USA
shafiulazam.chowdhury@mavs.uta.edu

Christoph Csallner
Computer Science & Eng. Dept.
University of Texas at Arlington
Arlington, TX 76019, USA
csallner@uta.edu

Abstract*—*Background: **Empirical studies on widely used model-based development tools such as MATLAB/Simulink are limited despite the tools' importance in various industries.**

Aims: **The aim of this paper is to investigate the reproducibility of previous empirical studies that used Simulink model corpora and to evaluate the generalizability of their results to a newer and larger corpus, including a comparison with proprietary models.**

Method: **The study reviews methodologies and data sources employed in prior Simulink model studies and replicates the previous analysis using SLNET. In addition, we propose a heuristic for determining code-generating Simulink models and assess the open-source models' similarity to proprietary models.**

Results: **Our analysis of SLNET confirms and contradicts earlier findings and highlights its potential as a valuable resource for model-based development research. We found that open-source Simulink models follow good modeling practices and contain models comparable in size and properties to proprietary models. We also collected and distribute 208 git repositories with over 9k commits, facilitating studies on model evolution.**

Conclusions: **The replication study offers actionable insights and lessons learned from the reproduction process, including valuable information on the generalizability of research findings based on earlier open-source corpora to the newer and larger SLNET corpus. The study sheds light on noteworthy attributes of SLNET, which is self-contained and redistributable.**

***Index Terms*—reproducibility, replication, Simulink, open science, code generation, Simulink model**

I. Introduction

There are only a few empirical studies of open-source MATLAB/Simulink artifacts, maybe due to a widespread perception that open-source Simulink artifacts are typically small, do not represent closed-source development, and are often hard to acquire [14], [15], [27], [61], [70]. Most empirical Simulink studies to date have instead relied on academic-industry collaborations—to get access to large closed-source Simulink artifacts [6]. Most empirical results on Simulink development and artifacts are thus based on case-studies of closed-source artifacts that (even when providing detailed experimental design descriptions and measurement tools) are hard to reproduce or replicate [10].

It is well-known how important replication is for scientific progress. Successful experiments need to be cross-validated under different conditions before they can be considered a part of science and interpreted with confidence [12]. Working towards large open-source Simulink corpora and empirical results that are easier to reproduce and replicate are thus important goals, given how widely Simulink is used in industry in safety-critical domains such as automotive and healthcare.

Towards these goals, recent initial work created via manual mining a first large corpus (which we call SC [17]) of open-source Simulink models and investigated modeling practices on a re-collected version of that corpus (SC_{20} [7]). The work found that some of these manually-collected Simulink models are suitable for empirical research, based on model metrics analysis and a qualitative assessments by a domain expert [7]. Follow-up work automated Simulink model collection, yielding the larger SLNET corpus that also allows redistribution [88]. However we are not aware of earlier work that either characterizes this larger SLNET corpus or uses it to replicate earlier empirical studies of Simulink models.

We thus first reproduce studies that are based on the initial SC large-scale Simulink model corpus, identifying inconsistencies in the original studies. We then replicate results of the earlier studies using the newer and larger SLNET corpus. By re-running the original study designs, we found inconsistencies between the experimental results and the ones reported in the paper, attributable to oversight and incomplete documentation. Our replication study using SLNET confirmed several previous findings, such as the low utilization of model references and algebraic loops. In contrast to prior work, we only found a weak correlation between cyclomatic complexity and other model metrics. To summarize, this paper makes the following major contributions.

- Through empirical data, we identify inconsistencies in earlier empirical Simulink studies.
- We characterize the SLNET corpus in relation to earlier corpora of open-source Simulink models.
- On SLNET we replicate previous studies, which both confirms and contradicts earlier findings.
- We collect and distribute 208 SLNET git repositories, containing 9k+ commits including 5k model versions, as artifacts that can be analyzed by the community [85].
- Our analysis tool [82] as well as reproduction and replication data [85] are open-sourced and available.

978-1-6654-5223-6/23$31.00

II. Background

Using Simulink's graphical modeling environment, engineers can design a complex system model as a hierarchical *block diagram* [50]. Each block represents a dynamic system that may take input through its *input ports* and produce output via its *output ports*, either continuously or at specific points in time. A block can be from a Simulink built-in library [53], from a separate *toolbox* library, or a custom *S-function* block defined via "native" code (e.g., in C). Blocks pass data to each other via directed *connections* (aka lines). Simulink is a commercial de-facto standard tool-chain in several domains such as aerospace, automotive, healthcare, and industrial automation.

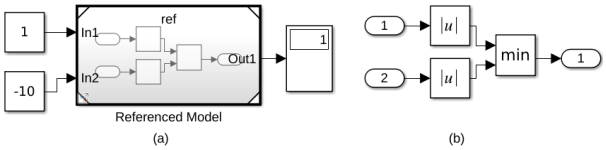


Fig. 1: (a) A tiny Simulink example model, (b) shows the contents of (a)'s referenced model.

Simulink offers several hierarchy mechanisms, ranging from a *subsystem* block grouping that can only be used in one context to a *model reference* (which essentially calls an independent model via its own well-defined interface and can thus be widely reused) [39]. These constructs allow further recursive decomposition, enabling deeply nested models. Figure 1(a) shows a tiny example hierarchical model that contains a model reference to the Figure 1(b) referenced model. Alternatively, the user can use library-linked blocks [49], that are references to blocks defined in a custom library [41], that enables reusability and centralized maintenance of block functionality across multiple models.

A *compiled* model can be simulated, where Simulink successively computes the output of each block over a specified time range using pre-configured numerical *fixed-step* and *variable-step* solvers. In an *algebraic loop*, a block's output can reach its input port in the same simulation step (i.e., without passing through a delay block), which complicates simulation. Besides *normal* mode, Simulink offers various *accelerator* modes to speed up simulation [48]. With additional toolboxes [37], from the model the user can then generate and deploy low-level code to the target hardware.

A. Simulink Modeling Guidelines and Best Practices

The MathWorks Advisory Board (MAB) is a group of commercial MathWorks customers that (starting with Daimler, Ford, and Toyota in 2001) publishes guidelines and best practices on developing and maintaining Simulink models. Besides standardization, these guidelines address key software engineering challenges such as creating models that are well-defined, readable, easy to integrate, and reusable.

In their current 2020 version [35] these guidelines include to (1) avoid algebraic loops as they are hard to simulate and cannot be compiled to target hardware, (2) use S-functions to implement custom algorithms, (3) use subsystems to modularize the model by functional decomposition, and (4) use model references to create hierarchies of reusable components.

B. Cyclomatic Complexity & Size Metrics in Simulink

McCabe introduced cyclomatic complexity and argued it corresponds to our intuitive notion of complexity. McCabe also reported on a set of 24 Fortran subroutines with "high" ($>$10) cyclomatic complexity. The subroutines' ranking by cyclomatic complexity closely correlated with their ranking by reliability [64]. With some 9k citations this article has been highly influential in academic software engineering.

Some five decades later the question of measuring program complexity and program understanding remains an active research area with several recent advances [18], [21], [31]. Researchers keep returning to cyclomatic complexity with recent tweaks [5], [13] and more fine-grained measures [2]. An example controversy was if cyclomatic complexity is just a proxy for program size (e.g., lines of code in a Java- or C-like language) [26], with recent empirical data showing cyclomatic complexity to remain independently valuable [30].

For Simulink, recent work has shown the value of size metrics (i.e., block count), e.g, metric outliers yield interesting findings [76]. Such results are also eventually reflected in industry practices. For example, while the MAB industry board's 2001 Simulink guidelines did not yet mention size metrics, the current 2020 version contains a recommendation ($\leq$60 LOC / function) [35]. However neither MAB guideline version mentions McCabe or cyclomatic complexity yet.

For calculating a metric, Simulink basically first flattens a given model into a single hierarchy level, essentially "inlining" both subsystems and referenced models. So if two blocks in a model refer to the same referenced model, for metric calculations the referenced model will appear in the flattened model twice. Simulink has an option to also similarly (recursively) inline the contents of (any) library blocks and prior work is split on activating this option when reporting metric results.

While a block diagram does not represent a procedural language's control-flow graph, Simulink still has several block types that provide control-like functionality. For example, the value a multiport-switch block receives on its first input port selects which of the remaining input ports the block will forward to its output port [51] (which corresponds to a procedural switch or nested if construct). Simulink thus first defines the cyclomatic complexity of each built-in block as the number of the block's conceptual branching decisions (i.e., mostly zero or one) and then sums up the cyclomatic complexity of all blocks in a given (flattened) model [42].

C. Scope of Empirical Studies of Simulink Models

The limited availability of repositories with large numbers of freely accessible Simulink models has restricted empirical studies that seek to understand Simulink model characteristics and metrics [4], [23], [67]. For example, Dajsuren et al. [19] investigated model metrics including cohesion and coupling using small subset of Simulink models.

Open-source Simulink models are generally considered insufficient to meet the high industry standards required for meaningful results [10]. To address this issue, Altinger et al. [3] published metrics from three proprietary Simulink models for researchers to analyze. However, the dataset is no longer available. Schroeder et al. studied 65 proprietary automotive Simulink models and found via interviews that engineers preferred simple size metrics such as block count over structural metrics to capture model complexity [75].

D. SC: First Corpus of Open-Source Simulink Models

Via a two-stage process Chowdhury et al. created what we call SC, the first corpus of freely available Simulink models [14], [17]. First [14], the research team collected 391 models, i.e., 41 of the MathWorks's tutorial models the team considered to not be "toy" examples, the open-source models from MATLAB Central that were most popular (by ratings or downloads), GitHub keyword search results, and 28 models from academic papers, colleagues, and Google searches. Second, the team added the Simulink models of 12 SourceForge repositories and of the 96 most-downloaded MATLAB Central projects, yielding a study of a total of 1,071 Simulink models [17].

SC classifies its 1,071 models as tutorial (41), simple (442), advanced (452), and other (136). The distinction between simple and advanced is determined heuristically: any GitHub project with forks or stars and any MATLAB Central project that are not academic assignment are labeled "Advanced". Models shipped with MATLAB/Simulink are labeled "Tutorial", while models from other sources are labeled 'Other'.

Overall, SC collects Simulink models of projects that (at least partially) are selected and labeled manually. While initially "only" providing project URLs [14], the full corpus [17] includes Simulink model files, metadata, and collection tools and is stored on a Google Drive directory linked from the project's GitHub homepage.

Analyzing the corpus with Simulink R2017a, the work found good modeling practices such as model referencing were not widely used. The work found MathWorks's cyclomatic complexity to be at most moderately correlated[1] with various other model metrics. The correlation was strongest (0.55) for the model's maximum hierarchy depth, followed by the model's number of contained subsystems (NCS). This contrasted with an earlier study by Olszewska et al. [67], which showed strong (0.73) correlation between MathWorks's cyclomatic complexity and the model's number of contained subsystem (NCS).

E. SC_{20}: SC Projects Recollected in 2020

In August 2020—some three years after SC was published [17], Boll et al. (a research team acting independently of Chowdhury et al.) collected what we call SC_{20} [7], i.e., the latest Simulink model versions of SC's Simulink projects, yielding 1,734 Simulink models. Simulink models, metadata, and the team's collection tools are preserved on Figshare [8].

The work evaluated SC_{20}'s suitability for empirical model-based research, analyzing each SC_{20} project's domain, origin, and model metrics. The work also proposed a heuristic for identifying models configured for code generation. The paper's analysis found that the majority of SC_{20} models were inadequate for most empirical research, but identified a few mature models. The work also noted that some SC_{20} GitHub projects' characteristics (e.g., a high number of commits and collaborators) suggest potential for evolution research.

F. SLNET: Largest Known Simulink Corpus

In February 2020 Shrestha et al. collected the SLNET corpus [88], which addresses key issues of SC and SC_{20} (i.e., manual project selection and unclear project licenses), yielding the first redistributable corpus of open-source Simulink models. Specifically, SLNET collects Simulink projects from the GitHub API and from MATLAB Central's RSS feed and does not include projects without Simulink model files, known model generators and their synthetic models, projects that do not have an appropriate license, potentially duplicate projects (via bijection of the projects' models' metrics), and projects whose models all have zero blocks, yielding 9,117 Simulink models. Simulink models, metadata, and the team's collection tools are preserved on Zenodo [80], [81], [87].

Combining models from the two largest collections of open-source Simulink models (GitHub and MATLAB Central), SLNET is 8 times larger than the largest previous corpus of Simulink models (SC). In March 2023 we confirmed that other hosting sites (still) contain significantly fewer public Simulink repositories (i.e., we could only find 52 Simulink projects on SourceForge and one on GitLab).

III. Research Design

Our goal is to gain a deeper understanding of the reproducibility and replicability in model-based development research, particularly regarding Simulink models, as emphasized in a recent literature review [10]. The literature review identified a single study that conducted a large-scale empirical investigation, emphasizing open science, i.e., SC [17]. Subsequently, members of the literature review team undertook their own investigation, by collecting the latest version of the models of the same corpus, i.e., SC_{20} [7].

The recently released SLNET corpus [88] has rectified limitations of the two existing corpora, allowing us to replicate the results of earlier empirical studies. Thus, we perform a sample study utilizing the existing corpora and employ a statistical learning strategy to generalize the findings of prior studies on a smaller dataset to a larger dataset [90], [91]. As such, our replication efforts serve a confirmatory purpose.

To structure our study effectively, we have formulated two primary research questions that center around reproducibility and replication.

[1]The earlier work discussed in this paragraph and our own analysis all use Kendall's τ at a 0.05 significance level and follow a recent labeling of subsequent $|\tau|$ ranges at that level, i.e.: "weak" below 0.4, then "moderate" to below 0.7, "strong" to below 0.9, etc. [34]

I What challenges and implications arise when attempting to reproduce model-based development research, specifically for Simulink models?

II To what extent can we generalize prior studies' findings to a dataset that is open-source or larger?

RQ1 In terms of basic Simulink model metrics, how does SLNET compare with earlier open-source corpora and what we know about industrial models?

RQ2 Is SLNET suitable for empirical studies of Simulink projects and their change histories?

RQ3 How do empirical results obtained on smaller open-source corpora and closed-source industry models carry over to the larger SLNET corpus?

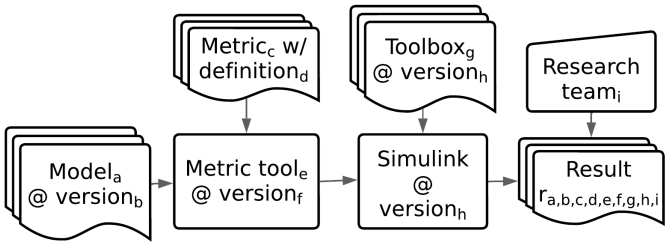


Fig. 2: Parameters *a* through *i* for reproducing and replicating results on Simulink models. Relative to earlier studies (and unless noted otherwise), for reproduction we only varied *i* and for replication we only varied *a,b,i*.

Figure 2 applies ACM's guidelines on reproducibility ("different team, same experimental setup") and replicability ("different team, different experimental setup") to empirical studies of Simulink models and summarizes the relevant variables. The following sections point out where we had to deviate from this model (e.g., when an exact earlier corpus is no longer available for exact reproduction).

IV. Corpora to Reproduce & Replicate Results

TABLE I: Overview of three existing (top) plus our four new or re-collected corpora (bottom) of open-source Simulink models; cut-off = date of latest model version in corpus; × = cannot distribute due to unclear licenses.

Corpus	Version of Simulink Models	Cut-off	Data
SC	Original corpus	2017	[77]
SLNET	Larger corpus	Feb '20	[87]
SC_{20}	SC re-collected at later version	Aug '20	[8]
SC_R	SC re-collected at SC's version	2017	×
SC_{20R}	SC_{20} completed at SC_{20}'s version	Aug '20	×
$SC_{20REvol}$	SC_{20R} GitHub projects' Git histories	Apr '23	×
$SLNET_{Evol}$	SLNET GitHub projects' Git histories	Apr '23	[85]

Table I summarizes the corpora of this study. Boll et al. [7] highlighted that the SC study results had several inconsistencies and Shrestha et al. [88] claimed earlier corpora suffer from unintended human errors and bias. Since both claims lacked sufficient empirical evidence, we attempted to reproduce these studies.

A. SC_R Corpus to Reproduce SC Results

To reproduce the SC study results, we downloaded all models and metadata from SC's Google Drive [77], yielding 1,347 models. This did not include all of the original study's 1,071 models, as the SC distribution excludes 169 models for their unclear licenses. We use SC's source metadata (for 862 of 1,071 models SC lists project URL and version, models studied within the project, and MATLAB version requirements) and retrieve 142 of 169 of these unclearly-licensed models from GitHub (at the same version as in SC).

For 40 of 1,071 models the download included multiple model versions but the metadata did not specify which version was used in the SC study. Since SC only provides aggregated model metrics (instead of per-model measurements), we could not disambiguate same-name models via comparing the metrics. After receiving confirmation from the SC team, we add all 113 potential model name matches from the SC download, yielding 1,117 models in SC_R (but still missing 27 of 1,071 now inaccessible GitHub models).

Due to the above model name ambiguity (or human error in SC creation), 5 of 1,071 models are now categorized as both Simple and Advanced. Since the SC study reported results per model category, we focused our reproduction on the one category not affected by the above missing/duplicate model issues, i.e., the 41 models labeled "tutorial". Since these 41 models ship with Simulink and we have access to earlier Simulink releases, it was straight-forward to reproduce the SC study results on the same version of the same models on the same Simulink version as the SC study.

The SC paper states that some reported metric results come from a third-party tool [72]. But we found the tutorial models' reported metrics instead exactly match the results of only running the SC metrics tool (which calls the Simulink API [54]). Specifically, we ran the SLNET-Metrics tool [80] as it can run SC's metric tool in the Simulink toolbox configuration [78] the SC study used, yielding the reported 10,926 blocks (as opposed to 10,391 the other tool returns [72]). After this calibration on the tutorial models we ran the SC tool in the same configuration on the rest of SC_R.

Finally, we clarified with the SC team SC's "S-function reuse rate", which SC defined to approximate how often a model contains an S-function it contains elsewhere. The metric basically counts how many S-function blocks in a model have the same name. For example, if a model contains four S-function blocks, three named "a" and one named "b", the reuse rate would be (3-1 + 1-1)/(3+1) = 0.5. SC reported a median reuse rate below 0.5%. Our result on SC_R being much higher triggered an interaction, in which the SC team confirmed that the SC paper mistakenly added the percentage symbol.

B. SC_{20R} & $SC_{20REvol}$ Corpora to Reproduce SC_{20} Results

We obtained the SC_{20} replication package (v2) from Figshare [8], which contains 1,736 models grouped into 194 projects with non-model files removed. The SC_{20} team categorized projects into four groups based on affiliation: 112 academic, 34 industry-mathworks, 25 industry, and 23

no-information. We included one project with an unknown category in the 'no-information' category, yielding SC_{20R}.

To extract model metrics, SC and SC_{20} mostly use the Simulink API, but there are differences. For instance, SC_{20} counts blocks via sldiagnostics [57] while SC uses Simulink Check [54] (the counts can differ). Additionally, SC uses the Simulink API for cyclomatic complexity, while SC_{20} implements McCabe's definition (independent paths). From the SC_{20} paper [7] and our correspondence with the SC_{20} team we could not reconstruct how SC_{20} computed project-level cyclomatic complexity.

The remaining model metrics we reproduced using the provided tool and documentation. To run the tool we had to install Simulink R2020a and the Check toolbox. We observed discrepancies in the results of 11/1736 models, which we attribute to a lack of documentation regarding the exact Simulink configuration (i.e., toolbox, library, etc).

The SC_{20} team analyzed 35 GitHub projects, but didn't include the necessary git repositories or commit extraction tool in the replication package. We independently developed the tool, and after contacting the authors, they updated their package, but the repositories remained missing. In April 2023 via metadata we obtained 32/35 repositories ("$SC_{20REvol}$"). 3/35 repositories were no longer online.

> ***Finding 1:*** *The SC and SC_{20} replication packages are insufficient to reproduce the original studies' results.*
> ***Implication:*** *Authors should host the replication package in permanent archival repositories for long-term access and preservation with documentation, such as Simulink configuration and instruction [65].*

C. SLNET Results & $SLNET_{Evol}$ Corpus

While the SLNET paper does not present any specific study or analysis, it does offer a valuable resource in the form of a corpus of Simulink models along with associated metadata on their metrics. In our attempt to reproduce SLNET's metrics, we first downloaded their corpus from Zenodo [87], which consists of 225 GitHub and 2,612 MATLAB Central projects, as well as a SQLite database of metadata. Following their documentation on Simulink configuration [83], we ran SLNET-Metrics, SLNET's metric collection tool, first on R2018b and then R2019b, as the latter ignores 'resource' folder, which some older SLNET projects use. By following this process, we were able to reproduce their reported metrics.

Like SC_{20}, SLNET only offers project snapshots, but to assess its suitability for evolution studies, we require its git repositories. In April 2023 we obtained 208/225 SLNET GitHub repositories, as 17 projects were offline. We refer to this collection as $SLNET_{Evol}$, which we have made available for other researchers to analyze.

D. Issues in Simulink Tool-chain Found

While trying to reproduce SLNET's results, we encountered the following two Simulink issues. MathWorks classified the first one as a bug and the second one as a documentation issue. First, when using multiple machines to speed up metric collection, Simulink R2018b crashed while compiling a SLNET model on Windows but compiled the model without issue on Ubuntu. We reported this issue (#04254318), which MathWorks confirmed as a bug and fixed in Simulink R2021b.

Finally, we reported (#04386513) that the cyclomatic complexity definition of the multiport switch [51] did not seem to match Simulink's metrics results. MathWorks addressed this issue by updating its public metric description [42].

V. Replicating Empirical Results Using SLNET

To date, empirical data on Simulink models and projects have been obtained on select closed-source projects and smaller open-source corpora (i.e., SC and SC_{20}). We would thus like to know how these earlier results generalize to the larger SLNET corpus of 2,837 open-source projects and their 9,117 Simulink models. As earlier work has not characterized SLNET, we will first put it into context for any subsequent findings or comparisons.

As in similar comparative studies, when interpreting experimental results we need to know how much results are skewed by differences in experimental setups. While conceptually straight-forward, calculating Simulink metrics is influenced by many parameters (Figure 2) and we realized that earlier studies did not document all relevant parameter values.

To increase confidence in our results we replicate earlier experiments where possible. Unless noted differently we apply the same metric extraction setup to all corpora—i.e., the same of our researchers use a single consistent set of metric definitions, metric tool version (SLNET-Metrics), Simulink version (R2020b on Ubuntu 18.04), and toolboxes [84].

We used Simulink R2020b as it enhanced metric calculation [45]. For example, in Simulink R2019b a video surveillance system's [24] cyclomatic complexity is 38,403, which on manual inspection seems highly inflated. For the same system Simulink R2020b returns 322. Such a drastic change makes it hard to directly compare our results with results reported elsewhere, e.g., the SLNET work used Simulink R2019b [88].

> ***Finding 2:*** *Small changes in experimental setup can drastically skew Simulink model metrics. In one example, upgrading to a newer version of Simulink changed a model's cyclomatic complexity from 38,403 to 322.*
> ***Implication:*** *There are subtle but severe pitfalls when comparing Simulink metric results across papers. To increase confidence in such comparisons we thus repeat earlier experiments where possible.*

A. Removing User-defined Libraries And Test Harnesses

User-defined libraries and test harnesses serve different goals than regular Simulink models. As they are also structurally different, we first identify and separate them from the regular models. While user-defined libraries are interesting themselves, for analyzing regular models we treat user-defined libraries like all other libraries. We thus either inline blocks from all or none of the libraries. Following prior work [17],

we use the Simulink API [38] and identify 235 user-defined libraries in SC_R, 411 in SC_{20R}, and 1,137 in SLNET.

Simulink's Test API [56] can identify models as test harnesses and we thus remove 9 test harnesses from SLNET and two each from SC_R and SC_{20R}. This is likely an undercount, as many open-source projects may not have the license necessary for this API and thus use workarounds. We thus heuristically label (but not remove) models as potential test harnesses by checking if model and folder names contain "test" or "harness", thereby labeling 143 models in SC_R, 233 in SC_{20R}, and 903 in SLNET.

B. RQ1: Basic Simulink Model Metrics of Corpora

At a high level, while it contains significantly more models, SLNET is not a superset of the previous open-source corpora. Even when containing the same model, corpora may differ in the included model version, due to different corpus collection times. When treating all versions of a model as the same model and including user-defined library models, SLNET contains 30% of the SC models $(328/1071)$, 36% for SC_R $(402/1117)$, 28% for SC_{20} $(492/1734)$, and 28% for SC_{20R} $(492/1736)$.

The remainder of this work removes from each corpus each model that is a test harness or a user-defined library. This differs from earlier work that treated user-defined library models as regular models and thus included them in overall metric counts [7]. (The only exceptions are the three Table II $\sqcup^{t0}$ columns, which inline user-defined libraries.) Table II compares SC_R, SC_{20R}, and SLNET on basic Simulink model metrics, such as number of models, models that are hierarchical, blocks, connections, and solver and simulation modes.

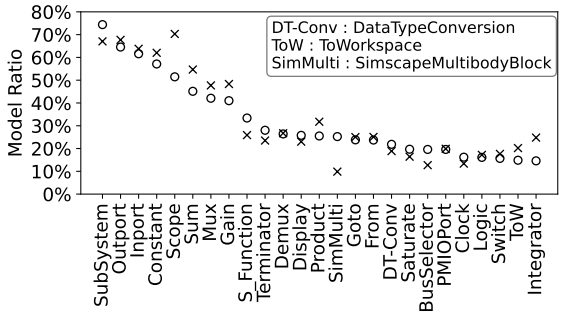
1) Model Size: A widely-used proxy for model size is the model's number of blocks [25], [63], [74]. For example, a recent paper conducted experiments on what it introduced as large industrial automotive models, containing 3.7k–73k blocks (and having hierarchy depth 8–16) [68]. Boll et al. report conversations with Simulink experts indicating typical industrial models often have 1k–10k blocks [10]. Industry-scale models at automotive supplier Delphi were earlier reported to have on average some 750 blocks [32].

Table II shows that (except for "Others"), including imported library blocks (B^{t0}) at least doubles the overall block count. Focusing on 1k+ block models, SC's custom tool (which includes imported library blocks) found 93 such models in SC on Simulink R2017a. On Simulink 2020b, SC's tool found 132 such models in SC_R, 139 in SC_{20R}, and 799 in SLNET. When excluding any imported library blocks, SC_R contains 14 such models, SC_{20R} 15, and SLNET 148.

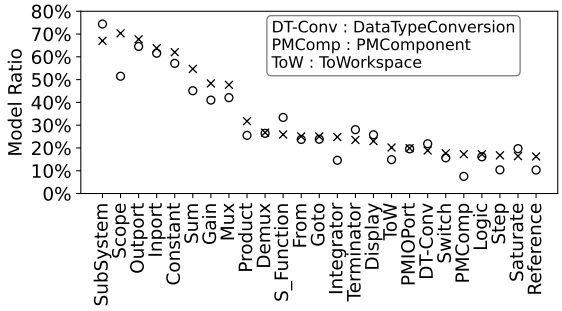
2) Hierarchical & Compiling Models: Model hierarchy is important for studying model complexity, model slicing and evaluating Simulink model generation tools [15], [16], [71], [89]. SC_R has 777 hierarchical models, of which we could compile 44%. Of SC_{20R}'s 852 hierarchical models we could only compile 20%. Of SLNET's 4.7k hierarchical models we could compile 47%. SC_{20R}'s low compile rate can be attributed to that corpus not distributing non-model files, which may have served as dependencies for the Simulink model.

3) Project and Model Metric Distributions: Table III shows model metric distributions across SC_R, SC_{20R}, and SLNET. The majority of SLNET models are relatively small, with mean exceeding median values. The overall distribution of metrics in SLNET is akin to that of earlier corpora, i.e., offering a broad spectrum with most standard deviations exceeding the means. SLNET however offers a broader range of Simulink models with similar min but notably larger max metric values. Following are additional distribution details of project size, most frequently used block types, and file types.

a) Project size: Similar to earlier corpora, the distribution of models in SLNET is skewed towards a few large projects. The 50 largest projects (i.e., the largest 1.8% of projects) contain 35% of all models, while 76% of the projects contain just one model. Some SLNET projects feature 18 empty models alongside non-empty models. By comparison, in SC_{20R}, 5/194 projects contain 35% of the models, and 53% of the projects contain just one model. With the exception of a single SLNET project that comprises a library model, all projects include some blocks and signal lines.



(a) Most-common block types in SC_R (o) and their SLNET (x) rate.



(b) Most-common block types in SLNET (x) and their SC_R (o) rate.

Fig. 3: Most-common block types in SC_R (a) and SLNET (b).

b) Most Frequently Used Block Types: Figures 3a and 3b show that the distributions of the most-commonly used block types are similar in SC_R and SLNET. For example, in each corpus over 60% of models contain a SubSystem block, making SubSystem appear in the most models in both corpora. SLNET uses SubSystem less-widely, likely as 28% of SLNET models have less than 8 blocks, which typically does not require a SubSystem block.

SC_R models use 156 distinct block types vs. 203 in SLNET (150 are in both). SLNET thus offers a potentially valuable

TABLE II: Model metrics after removing library & test harness models in SC_R (top), SC_{20R} (middle), and SLNET (bottom); M = models; Mc = models compiling in our setup; Mh = hierarchical models; C = non-hidden connections; $\lrcorner^{t0}$ = via SC's metric tool; var = variable; nor = normal; ext = external; PIL = processor in the loop; ac = accelerator; rap = rapid accelerator; Industry-M = Industry Mathworks; M-Central = MATLAB Central; excludes 14 SLNET models that crash Simulink R2020b; includes 20 SLNET models for which Simulink R2020b does not show solver and simulation metrics.

	Models		Hierarchical		Blocks		Connections		Solver Step		Simulation Mode				
	M	Mc	Mh	Mh^{t0}	B	B^{t0}	C	C^{t0}	fixed	var	nor	ext	PIL	ac	rap
Tutorial	41	41	37	40	3,703	13,917	3,700	14,020	13	28	41	0	0	0	0
GitHub	165	92	53	151	7,350	20,734	7,967	21,500	60	105	162	2	0	1	0
M-Central	674	294	488	595	76,473	483,645	80,683	473,466	257	417	655	14	1	4	0
SourceForge	230	33	196	201	18,444	126,123	17,800	125,021	183	47	175	55	0	0	0
Other	7	4	3	7	611	680	636	701	1	6	7	0	0	0	0
$\sum$ SC_R	1,117	464	777	994	106,581	645,099	110,786	634,708	514	603	1,040	71	1	5	0
Academic	690	232	456	634	75,813	185,574	86,223	185,733	229	461	597	68	0	16	9
Industry-M	404	61	259	351	30,826	220,011	27,631	212,299	176	228	399	4	1	0	0
Industry	174	15	93	161	24,753	180,929	25,116	194,655	135	39	169	3	0	1	1
No info	55	24	44	46	4,889	26,690	5,524	26,803	29	26	54	1	0	0	0
$\sum$ SC_{20R}	1,323	332	852	1,192	136,281	613,204	144,494	619,490	569	754	1,219	76	1	17	10
GitHub	1,637	541	875	1,297	190,213	424,175	188,069	400,753	860	759	1,498	103	2	14	2
M-Central	6,239	3,370	3,874	5,485	828,210	3,197,090	914,857	3,074,782	1,753	4,484	5,971	186	2	76	2
$\sum$ SLNET	7,876	3,911	4,749	6,782	1,018,423	3,621,265	1,102,926	3,475,535	2,613	5,243	7,469	289	4	90	4

TABLE III: Model (after removing library & test harness models) metric distributions per project (p) and per model (m) in SC_R (R), SC_{20R} (20R), and SLNET (N); Cyclom. C. = cyclomatic complexity (for a project the max of its models); Model Ref. = model references; Alg. L. = algebraic loops; LL Blocks = library linked blocks; Sub. Blocks = blocks in a subsystem at depth that has most such blocks.

		Min			Max			Average			Median			Standard Deviation		
		R	20R	N	R	20R	N	R	20R	N	R	20R	N	R	20R	N
Models	p	1	1	1	124	124	237	5.6	6.9	2.8	1.0	1.0	1.0	14.7	16.4	9.7
Blocks	p	1	1	0	13,555	13,831	172,196	457.4	706.1	362.8	116.0	140.0	52.0	1,419.9	1,959.8	3,577.1
	m	1	0	0	13,555	13,555	18,255	95.4	103.0	129.3	25.0	25.0	27.0	448.4	430.6	690.1
Block types	p	1	1	1	55	58	104	18.3	19.2	13.2	16.0	17.0	11.0	11.1	11.9	9.3
	m	1	1	1	47	47	101	10.6	10.4	10.4	8.0	8.0	8.0	8.3	7.8	7.9
Connections	p	0	0	0	14,169	16,491	231,672	475.5	748.7	392.9	124.0	153.0	57.0	1,422.1	2,103.5	4,611.7
	m	0	0	0	14,169	14,169	25,078	99.2	109.2	140.0	26.0	27.0	28.0	466.2	453.7	887.1
Subsystems	p	0	0	0	1,809	1,873	19,622	46.9	68.4	34.0	7.0	7.0	2.0	179.3	210.8	414.1
	m	0	0	0	1,294	1,294	2,117	9.8	10.0	12.1	3.0	2.0	2.0	44.1	41.8	75.3
Cyclom. C.	p	0	0	0	322	322	2,404	27.7	30.7	22.2	7.0	7.0	5.0	49.4	54.1	81.1
	m	0	0	0	322	322	2,404	14.0	13.6	13.7	4.0	4.5	2.0	32.4	31.6	59.0
Model Ref.	p	0	0	0	4	10	54	0.1	0.1	0.1	0.0	0.0	0.0	0.4	0.8	1.5
	m	0	0	0	4	2	12	0.0	0.0	0.0	0.0	0.0	0.0	0.2	0.1	0.4
Alg. L.	p	0	0	0	7	9	37	0.2	0.2	0.1	0.0	0.0	0.0	0.7	1.0	1.1
	m	0	0	0	2	1	6	0.1	0.1	0.1	0.0	0.0	0.0	0.2	0.2	0.3
LL Blocks	p	0	0	0	657	423	2,311	9.1	11.8	5.6	0.0	0.0	0.0	53.8	48.7	81.1
	m	0	0	0	31	31	441	1.9	1.7	2.0	0.0	0.0	0.0	4.5	4.3	15.0
Sub. Blocks	-	2	2	3	21	21	100	9.6	9.5	9.1	11.0	11.0	7.0	3.9	3.9	11.5

resource for research studies [14], [73]. Both SC and SC_{20} studies included library-imported blocks and reported a lower occurrence of output blocks (e.g., Scope [52], Display [43], and ToWorkspace [59]) than SLNET. The possible explanation for this discrepancy is that, like in procedural programming languages (where programmers include logging statements at various execution points), libraries may not have such statements for efficiency purposes. This practice is also observed in Simulink modeling.

Furthermore, From [46] and Goto [47] blocks, which are typically used to improve the visual layout of the model, are equally widely used in SC and SLNET. However, excessive non-local usage of From and Goto blocks adversely affects readability and design, warranting further investigation.

c) File types: Each Simulink model is stored in one of two file formats, the MDL legacy file format or SLX. Introduced in Simulink R2012a, SLX conforms to the Open Pack-

aging Conventions (OPC) interoperability standard. Across corpora, few projects contain both MDL and SLX files (SC_R 3%, SC_{20R} 7%, and SLNET 2%). Overall the major file type has shifted from MDL in SC_R to SLX in SLNET (39% of SC_R models are in SLX, 45% for SC_{20R}, and 55% for SLNET). The prevalence of SLX files in open-source models is significant for developing SLX to MDL back-transformation tools [1].

In summary, SLNET shares many similarities with prior corpora and offers a broader view of open-source Simulink projects. The majority of SLNET models are small, which may be relevant for analyzing simple models [69], [79], [86], [89], while also including a substantial number of non-trivial models using diverse features.

Finding 3: *As in many other kinds of open-source projects [28], [33], SLNET project and model metrics follow long-tailed distributions.*
Implication: *Research studies may use SLNET subsets based on their objectives. The diverse SLNET corpus can help address generalizability challenges in model-based development research.*

VI. Replicating Findings on Modeling Practices

A. Converging Result: Model Referencing

Analogous to classes in object-oriented programming, model references [36] enable modular model design, unit testing, and code reuse. But similar to the SC work [17], we found that only 10 SC_R (0.9%), 18 SC_{20R} (1.4%), and 139 SLNET models (1.8%) use model referencing. Even when accounting for the skewed SLNET model size distribution, Table III shows that model reference use remains sparse.

B. Converging Result: Algebraic Loops

An algebraic loop arises from a circular dependency between a block's output and input at the same simulation time step. An algebraic loop may reduce simulation performance or prevent the solver from resolving the loop. As the SC work [17], we found such loops relatively rarely, with only 20 SC_R and 186 SLNET models containing such loops.

C. Converging Result: Small Class Phenomenon

Zhang et al. observed the "small class" phenomenon in Java programs (most classes have few lines of code while a few classes are large) and found a high correlation between class size and number of defects [92], [93]. In Simulink, subsystems are used to encapsulate a function, resulting in a hierarchical model. Similar to the small class phenomenon noted in the SC work [17], we observe that the median number of blocks in a subsystem at any hierarchy does not exceed 11 in both SC_R and SLNET. This may inform future hypotheses on Simulink subsystem size and defects.

Finding 4: *The median number of blocks in a subsystem at any hierarchy level does not exceed 11.*
Implication: *More research is needed to assess how subsystem size impacts Simulink model quality.*

D. Converging Result: S-function Reuse Rate

TABLE IV: S-function per-model reuse rate for models with 1+ S-functions; $M_{S\text{-}fct}$ = models with 1+ S-functions; LQ = lower quartile; UQ = upper quartile; med = median.

	$M_{S\text{-}fct}$	min	LQ	med	UQ	max	avg
SC_R	351	0.0	0.0	0.0	0.38	0.92	0.20
SC_{20R}	378	0.0	0.0	0.0	0.50	0.98	0.23
SLNET	1,504	0.0	0.0	0.0	0.50	0.99	0.21

Besides reuse of legacy C code, S-functions allow within-model code reuse (i.e., defined once but added to and used in several model components). In the same spirit as the SC work [17], Table IV shows that S-functions are not widely used, with just 31% of SC_R models and 20% of SLNET using S-functions. For models that use S-functions, 41% of SC_R models and 40% of SLNET models reuse at least one S-function (but these models' median S-function reuse rate is zero across corpora).

E. Diverging Result: Cyclomatic Complexity vs Other Metrics

We conduct a correlation analysis between cyclomatic complexity and the other Table V model metrics using Kendall's τ. We only use models for which we could calculate cyclomatic complexity (e.g., excluding models we could not compile). As in the SC study, for SC_R we used non-Simple models. For SC_{20R}, we used industry and industry-MathWorks models. As SLNET models are not categorized, we used those containing 200+ blocks. All metrics exhibit a statistically significant correlation at a 0.05 significance level.

TABLE V: Correlation between cyclomatic complexity and model metrics; M,B,C from Table II: models, blocks, and non-hidden connections; UB = unique block types; MHD = max. hierarchy depth; CRB = child-model representing blocks i.e., model reference and subsystem; NCS = contained subsystems.

	M	B	C	UB	MHD	CRB	NCS
SC_R	160	0.29	0.32	0.31	**0.38**	0.28	0.29
SC_{20R}	58	0.16	0.16	0.20	0.31	**0.41**	**0.41**
$SLNET_{\geq 200}$	279	0.27	0.27	0.23	0.10	**0.28**	0.27
$SLNET_{200\text{-}300}$	111	-0.02	0.12	**0.16**	0.05	0.07	0.07

SC_R models have a weak positive correlation (0.28 to 0.38) between cyclomatic complexity and model metrics. For SC_{20R} models the correlation is positive and weak to (barely) moderate (0.16 to 0.41). For SLNET models with 200+ blocks the correlation is positive but remains weak (0.10 to 0.28).

Finding 5: *Contrary to previous work [67], cyclomatic complexity does not seem strongly correlated with other model metrics.*
Implication: *Similar to Java- and C-like languages, in Simulink cyclomatic complexity seems to remain an independently valuable metric.*

TABLE VI: $SLNET_{Evol}$ and $SC_{20REvol}$ per-model (m) and per-project (p) change metrics; Total commits, commits per day during project duration, merge commits ($\succ$), and commits of 1+ mdl/slx files (MS); commit authors and $commit_{MS}$ authors; med = median; std = standard deviation.

		$SC_{20REvol}$					$SLNET_{Evol}$				
		min	max	avg	med	std	min	max	avg	med	std
Commits	p	1	590	62.7	10.5	124.6	1	963	43.9	7.5	120.1
Commit / day	p	0	4	0.9	0.3	1.2	0	24	1.9	0.6	3.1
$Commits_{MS}$ [%]	p	0	100	38.8	26.8	31.2	1	100	31.4	25.0	23.5
$Commits_{\succ}$ [%]	p	0	17	2.7	0.0	4.7	0	40	3.2	0.0	6.7
$Updates_{MS}$	m	0	43	3.3	1.0	5.7	0	53	1.8	1.0	2.8
Authors	p	1	16	2.8	2.0	3.5	1	21	2.0	1.0	2.6
	m	1	3	1.1	1.0	0.4	1	8	1.3	1.0	0.7
$Authors_{MS}$ [%]	p	0	100	68.6	75.0	34.5	10	100	82.2	100.0	26.1

F. Converging Result: Suitability For Change Studies

To assess their applicability for Simulink model and project change studies, we analyzed $SC_{20REvol}$'s 32 and $SLNET_{Evol}$'s 208 git repositories (for $SLNET_{Evol}$ we only studied the commits until SLNET's February 2020 snapshot). Three projects (with 811 commits) were in both corpora.

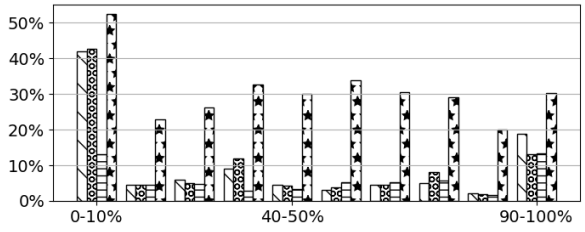
Table VI gives an overview of the project and model change metrics. For example, 53% of $SC_{20REvol}$ projects (17/32) and 39% of SLNET projects (82/208) are maintained by at least two collaborators, of which 8/17 and 32/82 have commits spanning over a year. Just 22% of $SC_{20REvol}$ and 15% of $SLNET_{Evol}$ projects have more than 50 commits. Across $SC_{20REvol}$ and $SLNET_{Evol}$ projects, 20% of commits involved updates or the creation of one or more models.

In both corpora, an average of 22% of models were under active development throughout the projects with 3+ commits, indicating the models were primary artifacts of these projects. However, 40% of $SC_{20REvol}$ and almost half of $SLNET_{Evol}$ projects did not update their models after committing them to the repository. In both corpora, roughly 55% of models were not updated at all. The lack of model updates may be due to GitHub Simulink projects mainly serving as archives—like most other GitHub projects [28].

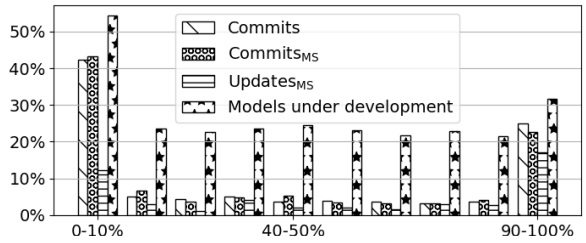
Figure 4 breaks each project's duration into 10 buckets of equal length (normalized to each project's duration). Here project duration is the duration from a project's first to last commit as recorded by the timestamps assigned by the authors' machines. While this approach has its pitfalls, the more-active projects are usually less affected and we performed the basic recommended sanity checks to ensure there are no impossible outliers (e.g., commits with Unix time zero) [22].

To avoid potential skewing caused by "code dump" projects, Figure 4 excludes projects with less than 3 commits, yielding 26 $SC_{20REvol}$ projects and 186 $SLNET_{Evol}$ projects. Even with this filtering, the figure may still be biased towards projects with fewer commits as the majority of both $SC_{20REvol}$ and $SLNET_{Evol}$ projects have less than 11 commits.

Finding 6: *A quarter of $SLNET_{Evol}$ projects are developed collaboratively and have 1+ multi-revision models.*
Implication: *$SLNET_{Evol}$ projects have the potential to yield valuable insight into open-source Simulink development.*



(a) Timeline of 26/32 $SC_{20REvol}$ 3+ commit projects.



(b) Timeline of 186/208 $SLNET_{Evol}$ 3+ commit projects.

Fig. 4: Across normalized project duration (x-axis): Total project commits, commits of 1+ mdl/slx files, individual mdl/slx file updates, and mdl/slx files under development (i.e., in between a file's first and last commit).

G. Diverging Result: Open-source Code Generation Models

Simulink models that can generate code are of interest in model-based research and tool-development [11], [29], [62], [63], [66]. Initially we applied SC_{20}'s heuristics to search for Embedded Coder [44] or TargetLink [20] traces. But we found inconsistencies between SC_{20}'s results (finding no code generation models) and their replication package's heuristics [9]. During our interactions the SC_{20} team acknowledged a bug and fixed it in their replication package version 2 [8].

Specifically, SC_{20}'s heuristics determine if a model can generate code based on the presence of *atomic* subsystems [60] or special TargetLink blocks. This found 33 SLNET models configured for Embedded Coder but no TargetLink traces. We found this heuristic restrictive and not specific to Embedded Coder. Our counter-example model had non-atomic subsystems and successfully generated code via Embedded Coder.

For background, while every Simulink model can generate code using Simulink Coder [55], this requires a fixed-step

TABLE VII: Models configured for code generation; M = all models; EC_{20} = SC_{20} Embedded Coder heuristics; EC = our Embedded Coder heuristics; GRT = Simulink Coder (Real-Time Workshop); Other = other code generation toolboxes.

	M	EC_{20}	EC	GRT	Other	Total
Tutorial	41	1	1	12	0	13
GitHub	165	0	4	52	4	60
MATC	674	0	47	101	109	257
Sourceforge	230	0	0	96	87	183
Others	7	0	0	1	0	1
$\sum$ SC_R	1,117	1	52	262	200	514
Academic	690	0	3	94	136	233
Industry-M	404	0	33	77	67	177
Industry	174	0	5	129	1	135
No Info	55	0	1	28	0	29
$\sum$ SC_{20R}	1,323	0	42	328	204	574
GitHub	1,637	14	129	502	234	865
MATC	6,239	19	423	1,050	297	1,770
$\sum$ SLNET	7,876	33	552	1,552	531	2,635

solver, which conflicts with the default variable-step solver model configuration. Simulink models further rely on a target language compiler (TLC) file [58] to map Simulink blocks and parameters to the target language's constructs.

Simulink offers a set of standard-named TLC files that support various solver types [40]. For example,'rsim.tlc' supports fixed-step and variable-step solvers. To determine if the Simulink model is configured to generate code, we follow a heuristic approach. First, we check if the model's TLC file name matches with one provided by Simulink and the model is configured with appropriate solver type. Second, in cases where the solver type required is ambiguous, we make a conservative assumption that the model must be configured with the fixed-step solver.

Table VII shows the number of models configured for code generation. SLNET has 2,635 models with code generation capabilities, at least 4× more than previous corpora.

> ***Finding 7:*** *SLNET has 4× models configured for code generation (a common configuration in industrial models) than the largest earlier open-source model collection.* ***Implication:*** *Additional investigation is required to determine if the code generation models in SLNET can meet requirements of research studies.*

VII. Threats to Validity

Internal validity concerns the experimental design, data collection and analysis. In our replication efforts, we closely adhered to the original study's setup and tools. We calibrated the provided tools and contacted the authors for clarification and consistency in data analysis. It is important to note that the choice of Simulink version can impact model metrics and introduce slight differences in insights.

Specifically, for a subset of 554 SLNET models (the models of the 10 SLNET projects with the most models) we compared model metrics obtained using both R2020b and R2022b. Results for all metrics were the same for all models, except for 3/554 models where the cyclomatic complexity differed by 2–6 between R2020b and R2022b.

External validity examines the generalizability of reproduced and replicated study results. In our case, the generalizability of our findings is limited to Simulink models within the SLNET corpus. SLNET may not represent all available Simulink projects, as its construction involved a keyword search on GitHub and filtering for redistributable projects. However, considering that the majority of results from the original studies, which involved some level of cherry picking in their corpus, hold true in SLNET–a larger dataset encompassing diverse models with a small overlap–we are optimistic in the generalizability of the presented results to other open-source Simulink models.

Construct validity ensures that the measures and metrics used in the replicated study accurately capture the intended concepts. Our confirmatory replication study inherits limitations from the original studies, such as not analyzing Stateflow blocks or MATLAB code, which can contribute to the project's complexity. Also, SC's heuristic used to identify test harnesses may have limitations, as manual inspection revealed 10% of such models are test harnesses. Upon noticing issues with SC_{20}'s code generation heuristic, we proposed new methods after consulting with the original authors.

Reliability refers to the replicability of a study for obtaining same or similar results. To mitigate reliability risks, we distribute our analysis tool and complete replication package as open-source via permanent storage locations [82], [85]. We encourage replication of our findings.

VIII. Conclusions and Future Work

The study investigated the reproducibility of previous empirical studies of Simulink models and evaluated the generalizability of their results to the larger SLNET corpus. The SLNET study confirmed and contradicted earlier findings, highlighting its potential as a valuable corpus for model-based development research and also provided actionable insights for future research. We found that open-source Simulink models generally follow good modeling practices and that few open-source models are comparable in size and properties to proprietary models. To that end, we proposed a heuristic to determine code generating Simulink models. We also provided 208 Git repositories to facilitate model evolution studies.

While this paper only analyzes Simulink model metrics focusing on reproducibility and replication, future work includes examining if the model metrics can be used to make predictions of process metrics such as defect prediction.

Acknowledgements

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] B. Adhikari, E. J. Rapos, and M. Stephan, "Simulink model transformation for backwards version compatibility," in *MODELS-C*. IEEE, Oct. 2021, pp. 427–436.

[2] S. Ajami, Y. Woodbridge, and D. G. Feitelson, "Syntax, predicates, idioms - what really affects code complexity?" *Empir. Softw. Eng.*, vol. 24, no. 1, pp. 287–328, Feb. 2019.

[3] H. Altinger, S. Siegl, Y. Dajsuren, and F. Wotawa, "A novel industry grade dataset for fault prediction based on model-driven developed automotive embedded software," in *MSR*. IEEE, May 2015, pp. 494–497.

[4] B. Balasubramaniam, H. Bagheri, S. Elbaum, and J. Bradley, "Investigating controller evolution and divergence through mining and mutation*," in *ICCPS*, 2020, pp. 151–161.

[5] M. M. Barón, M. Wyrich, and S. Wagner, "An empirical validation of cognitive complexity as a measure of source code understandability," in *ESEM*. ACM, Oct. 2020, pp. 5:1–5:12.

[6] V. Bertram, S. Maoz, J. O. Ringert, B. Rumpe, and M. von Wenckstern, "Component and connector views in practice: An experience report," in *MODELS*. IEEE Computer Society, September 2017, pp. 167–177.

[7] A. Boll, F. Brokhausen, T. Amorim, T. Kehrer, and A. Vogelsang, "Characteristics, potentials, and limitations of open-source Simulink projects for empirical research," *Software and Systems Modeling*, vol. 20, pp. 2111—-2130, Apr. 2021.

[8] A. Boll, T. Kehrer, A. Vogelsang, T. Amorim, and F. Brokhausen, "Characteristics, potentials, and limitations of open source Simulink projects for empirical research: Dataset," 2021. [Online]. Available: https://doi.org/10.6084/m9.figshare.13636589.v2

[9] ——, "Characteristics, potentials, and limitations of open source Simulink projects for empirical research: Dataset," 2021. [Online]. Available: https://doi.org/10.6084/m9.figshare.13636589.v1

[10] A. Boll, N. Vieregg, and T. Kehrer, "Replicability of experimental tool evaluations in model-based software and systems engineering with MATLAB/Simulink," *Innovations in Systems and Software Engineering*, pp. 1–16, Mar. 2022.

[11] H. Bourbouh, P.-L. Garoche, T. Loquen, É. Noulard, and C. Pagetti, "Cocosim, a code generation framework for control/command applications an overview of cocosim for multi-periodic discrete Simulink models," in *ERTS*, 2020.

[12] D. T. Campbell and J. C. Stanley, *Experimental and Quasi-Experimental Designs for Research*. Houghton Mifflin Company, 1963, p. 3.

[13] G. A. Campbell, "Cognitive complexity: An overview and evaluation," in *TechDebt*. ACM, May 2018, pp. 57–58.

[14] S. A. Chowdhury, S. Mohian, S. Mehra, S. Gawsane, T. T. Johnson, and C. Csallner, "Automatically finding bugs in a commercial cyber-physical system development tool chain with SLforge," in *ICSE*. ACM, May 2018, pp. 981–992.

[15] S. A. Chowdhury, S. L. Shrestha, T. T. Johnson, and C. Csallner, "SLEMI: Equivalence modulo input (EMI) based mutation of CPS models for finding compiler bugs in Simulink," in *ICSE*. ACM, May 2020, pp. 335–346.

[16] ——, "SLEMI: finding simulink compiler bugs through equivalence modulo input (EMI)," in *Proc. 42nd International Conference on Software Engineering (ICSE), Companion Volume*. ACM, May 2020, pp. 1–4. [Online]. Available: https://doi.org/10.1145/3377812.3382147

[17] S. A. Chowdhury, L. S. Varghese, S. Mohian, T. T. Johnson, and C. Csallner, "A curated corpus of Simulink models for model-based empirical studies," in *SEsCPS*. ACM, May 2018, pp. 45–48.

[18] S. A. Chowdhury, R. Holmes, A. Zaidman, and R. Kazman, "Revisiting the debate: Are code metrics useful for measuring maintenance effort?" *Empir. Softw. Eng.*, vol. 27, no. 6, Nov. 2022.

[19] Y. Dajsuren, M. G. J. van den Brand, A. Serebrenik, and S. A. Roubtsov, "Simulink models are also software: Modularity assessment," in *QoSA*, Jun. 2013, pp. 99–106.

[20] dSpace, "Targetlink," 2023, accessed June 2023. [Online]. Available: https://www.dspace.com/en/inc/home/products/sw/pcgs/targetlink.cfm

[21] D. G. Feitelson, "Considerations and pitfalls for reducing threats to the validity of controlled experiments on code comprehension," *Empir. Softw. Eng.*, vol. 27, no. 6, Nov. 2022.

[22] S. W. Flint, J. Chauhan, and R. Dyer, "Pitfalls and guidelines for using time-based Git data," *Empir. Softw. Eng.*, vol. 27, no. 7, pp. 1–55, Dec. 2022.

[23] W. Hu, T. Loeffler, and J. Wegener, "Quality model based on iso/iec 9126 for internal quality of matlab/simulink/stateflow models," in *ICIT*. IEEE, 2012, pp. 325–330.

[24] A. Hwang, "Video surveillance system design with Simulink and Xilinx FPGAs," 2022, accessed Nov 2022. [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/20160-video-surveillance-system-design-with-simulink-and-xilinx-fpgas

[25] M. Jaskolka, S. Scott, V. Pantelic, A. Wassyng, and M. Lawford, "Applying modular decomposition in Simulink," in *ISSRE-W*. IEEE, 2020.

[26] G. Jay, J. E. Hale, R. K. Smith, D. P. Hale, N. A. Kraft, and C. Ward, "Cyclomatic complexity and lines of code: Empirical evidence of a stable linear relationship," *J. Softw. Eng. Appl.*, vol. 2, no. 3, pp. 137–143, 2009.

[27] Z. Jiang, X. Wu, Z. Dong, and M. Mu, "Optimal test case generation for Simulink models using slicing," in *QRS-C*, July 2017, pp. 363–369.

[28] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. Germán, and D. E. Damian, "An in-depth study of the promises and perils of mining GitHub," *Empir. Softw. Eng.*, vol. 21, no. 5, pp. 2035–2071, 2016.

[29] J. Krizan, L. Ertl, M. Bradac, M. Jasansky, and A. Andreev, "Automatic code generation from MATLAB/Simulink for critical applications," in *CCECE*. IEEE, May 2014, pp. 1–6.

[30] D. Landman, A. Serebrenik, E. Bouwers, and J. J. Vinju, "Empirical analysis of the relationship between CC and SLOC in a large corpus of Java methods and C functions," *J. Softw. Eng. Appl.*, vol. 28, no. 7, pp. 589–618, Jul. 2016.

[31] O. Levy and D. G. Feitelson, "Understanding large-scale software systems - structure and flows," *Empir. Softw. Eng.*, vol. 26, no. 3, May 2021.

[32] B. Liu, Lucia, S. Nejati, and L. C. Briand, "Improving fault localization for Simulink models using search-based testing and prediction models," in *SANER*, Feb. 2017, pp. 359–370.

[33] W. Ma, L. Chen, Y. Zhou, and B. Xu, "What are the dominant projects in the github python ecosystem?" in *TSA*. IEEE, September 2016, pp. 87–95.

[34] M. A. A. Mamun, C. Berger, and J. Hansson, "Effects of measurements on correlations of software code metrics," *Empir. Softw. Eng.*, vol. 24, no. 4, pp. 2764–2818, 2019. [Online]. Available: https://doi.org/10.1007/s10664-019-09714-9

[35] MathWorks Advisory Board (MAB), "Control algorithm modeling guidelines using MATLAB, Simulink, and Stateflow," MathWorks Inc, Tech. Rep. Version 5.0, 2020. [Online]. Available: https://www.mathworks.com/solutions/mab-guidelines.html

[36] MathWorks Inc, "Model References," 2022, accessed Nov 2022. [Online]. Available: https://www.mathworks.com/help/simulink/model-reference.html

[37] ——, "Products and services," 2022, accessed Nov 2022. [Online]. Available: https://www.mathworks.com/products.html

[38] ——, "bdislibrary," 2023, accessed April 2023. [Online]. Available: https://www.mathworks.com/help/simulink/slref/bdislibrary.html

[39] ——, "Choose among types of model components," 2023, accessed April 2023. [Online]. Available: https://www.mathworks.com/help/simulink/ug/types-of-model-components.html

[40] ——, "Configure a system target file," 2023, accessed June 2023. [Online]. Available: https://www.mathworks.com/help/rtw/ug/select-a-target.html

[41] ——, "Custom libraries," 2023, June 2023. [Online]. Available: https://www.mathworks.com/help/simulink/libraries.html

[42] ——, "Cyclomatic complexity metric," 2023, accessed April 2023. [Online]. Available: https://www.mathworks.com/help/slcheck/ug/mathworks.metrics.cyclomaticcomplexity.html

[43] ——, "Display," 2023, accessed June 2023. [Online]. Available: https://www.mathworks.com/help/simulink/slref/display.html

[44] ——, "Embedded Coder," 2023, accessed June 2023. [Online]. Available: https://www.mathworks.com/products/embedded-coder.html

[45] ——, "Enhanced calculation of cyclomatic complexity," 2023, accessed February 2023. [Online]. Available: https://www.mathworks.com/help/slcheck/release-notes.html

[46] ——, "From," 2023, accessed June 2023. [Online]. Available: https://www.mathworks.com/help/simulink/slref/from.html

[47] ——, "Goto," 2023, accessed June 2023. [Online]. Available: https://www.mathworks.com/help/simulink/slref/goto.html

[48] ——, "How accelerator model works documentation," 2023, accessed June 2023. [Online]. Available: https://www.mathworks.com/help/simulink/ug/how-the-acceleration-modes-work.html

[49] ——, "Linked blocks," 2023, June 2023. [Online]. Available: https://www.mathworks.com/help/simulink/ug/creating-and-working-with-linked-blocks.html

[50] ——, "MATLAB & Simulink," 2023, accessed June 2023. [Online]. Available: https://www.mathworks.com/products/simulink.html/

[51] ——, "Multiport switch," 2023, accessed April 2023. [Online]. Available: https://www.mathworks.com/help/simulink/slref/multiportswitch.html

[52] ——, "Scope," 2023, accessed June 2023. [Online]. Available: https://www.mathworks.com/help/simulink/slref/scope.html

[53] ——, "Simulink block libraries documentation," 2023, accessed June 2023. [Online]. Available: https://www.mathworks.com/help/simulink/block-libraries.html

[54] ——, "Simulink Check," 2023, accessed June 2023. [Online]. Available: https://www.mathworks.com/help/slcheck/

[55] ——, "Simulink Coder," 2023, accessed February 2023. [Online]. Available: https://www.mathworks.com/products/simulink-coder.html

[56] ——, "Simulink Test," 2023, accessed April 2023. [Online]. Available: https://www.mathworks.com/help/sltest/

[57] ——, "sldiagnostic," 2023, accessed June 2023. [Online]. Available: https://www.mathworks.com/help/simulink/slref/sldiagnostics.html

[58] ——, "Target language compiler basics," 2023, accessed June 2023. [Online]. Available: https://www.mathworks.com/help/rtw/tlc/what-is-the-target-language-compiler.html

[59] ——, "To Workspace," 2023, accessed June 2023. [Online]. Available: https://www.mathworks.com/help/simulink/slref/toworkspace.html

[60] ——, "Treat as atomic unit," 2023, accessed April 2023. [Online]. Available: https://www.mathworks.com/help/simulink/slref/subsystem.html#brp1xt9-56

[61] R. Matinnejad, S. Nejati, L. C. Briand, and T. Bruckmann, "Effective test suites for mixed discrete-continuous stateflow controllers," in *ESEC/FSE*. ACM, Aug. 2015, pp. 84–95.

[62] ——, "Automated test suite generation for time-continuous Smulink models," in *ICSE*, May 2016, pp. 595–606.

[63] ——, "Test generation and test prioritization for Simulink models with dynamic behavior," *IEEE Trans. Software Eng.*, vol. 45, no. 9, pp. 919–944, 2019.

[64] T. J. McCabe, "A complexity measure," *IEEE Trans. Softw. Eng.*, vol. 2, no. 4, pp. 308–320, Dec. 1976.

[65] D. Méndez, D. Graziotin, S. Wagner, and H. Seibold, "Open science in software engineering," in *Contemporary Empirical Methods in Software Engineering*. Springer, 2020, pp. 477–501.

[66] M. M. R. Mozumdar, F. Gregoretti, L. Lavagno, L. Vanzago, and S. Olivieri, "A framework for modeling, simulation and automatic code generation of sensor network application," in *IEEE SECON*, 2008, pp. 515–522.

[67] M. Olszewska, Y. Dajsuren, H. Altinger, A. Serebrenik, M. A. Waldén, and M. G. J. van den Brand, "Tailoring complexity metrics for Simulink models," in *ECSA-W*, Nov. 2016, p. 5.

[68] V. Pantelic, S. M. Postma, M. Lawford, M. Jaskolka, B. Mackenzie, A. Korobkine, M. Bender, J. Ong, G. Marks, and A. Wassyng, "Software engineering practices and simulink: Bridging the gap," *STTT*, vol. 20, no. 1, pp. 95–117, 2018.

[69] V. Pantelic, S. M. Postma, M. Lawford, A. Korobkine, B. Mackenzie, J. Ong, and M. Bender, "A toolset for Simulink: Improving software engineering practices in development with Simulink," in *MODELS*. SciTePress, February 2015, pp. 50–61.

[70] A. C. Rao, A. Raouf, G. Dhadyalla, and V. Pasupuleti, "Mutation testing based evaluation of formal verification tools," in *DSA*. IEEE, Oct. 2017, pp. 1–7.

[71] R. Reicherdt and S. Glesner, "Slicing MATLAB Simulink models," in *ICSE*, June 2012, pp. 551–561.

[72] G. Rouleau, "How many blocks are in that model?" 2023, accessed June 2023. [Online]. Available: https://blogs.mathworks.com/simulink/2009/08/11/how-many-blocks-are-in-that-model

[73] B. Sánchez, A. Zolotas, H. H. Rodriguez, D. S. Kolovos, and R. F. Paige, "On-the-fly translation and execution of OCL-like queries on Simulink models," in *MODELS*. IEEE, 2019, pp. 205–215.

[74] A. Schlie, D. Wille, S. Schulze, L. Cleophas, and I. Schaefer, "Detecting variability in MATLAB/Simulink models: An industry-inspired technique and its evaluation," in *SPLC*, September 2017, pp. 215–224.

[75] J. Schroeder, C. Berger, T. Herpel, and M. Staron, "Comparing the applicability of complexity measurements for Simulink models during integration testing – an industrial case study," in *SAM*. IEEE, May 2015, pp. 35–40.

[76] J. Schroeder, C. Berger, M. Staron, T. Herpel, and A. Knauss, "Unveiling anomalies and their impact on software quality in model-based automotive software revisions with software metrics and domain experts," in *ISSTA*. ACM, Jul. 2016, pp. 154–164.

[77] Shafiul Azam Chowdhury, "Home," 2022, accessed Nov 2022. [Online]. Available: https://github.com/corpussimulink/corpus/wiki

[78] ——, "ICSE 2018 Artifacts," 2022, accessed Nov 2022. [Online]. Available: https://github.com/verivital/slsf_randgen/wiki/ICSE-2018-Artifacts

[79] S. L. Shrestha, "Automatic generation of Simulink models to find bugs in a cyber-physical system tool chain using deep learning," pp. 110–112, June 2020.

[80] ——, "50417/SLNET_Metrics: SLNET_Metrics MSR Release," Mar. 2022. [Online]. Available: https://doi.org/10.5281/zenodo.6336048

[81] ——, "SLNET-Miner," 2022, November 2022. [Online]. Available: https://github.com/50417/SLNet_Miner

[82] ——, "50417/SLReplicationTool: Replicability Study: Corpora For Understanding Simulink Models & Projects," Jul. 2023. [Online]. Available: https://doi.org/10.5281/zenodo.8111687

[83] ——, "MATLAB/Simulink Installation," 2023, accessed June 2023. [Online]. Available: https://github.com/50417/SLNET_Metrics/wiki/MATLAB-Simulink-Installation

[84] ——, "Simulink Model Version," 2023, July 2023. [Online]. Available: https://github.com/50417/SLReplicationTool/blob/main/MatlabInstallation.md

[85] S. L. Shrestha, S. A. Chowdhury, and C. Csallner, "Replicability study: Corpora for understanding simulink models & projects (analysis data) and slnet-evol dataset," Jul. 2023. [Online]. Available: https://figshare.com/articles/dataset/Replicability_Study_Corpora_For_Understanding_Simulink_Models_Projects/22064969

[86] ——, "DeepFuzzSL: Generating models with deep learning to find bugs in the Simulink toolchain," in *DeepTest*. ACM, May 2020, paper http://ranger.uta.edu/ csallner/papers/Shrestha20DeepFuzzSL.pdf.

[87] ——, "SLNET: A redistributable corpus of 3rd-party Simulink models: Dataset," Jun. 2020. [Online]. Available: https://doi.org/10.5281/zenodo.4898432

[88] ——, "SLNET: A Redistributable Corpus of 3rd-party Simulink models," in *MSR*. IEEE, May 2022, pp. 1–5.

[89] S. L. Shrestha and C. Csallner, "SLGPT: Using transfer learning to directly generate Simulink model files and find bugs in the Simulink toolchain," in *EASE*. ACM, 2021, pp. 260–265.

[90] K. Stol and B. Fitzgerald, "The ABC of software engineering research," *ACM Transactions on Software Engineering and Methodology*, vol. 27, no. 3, pp. 11:1–11:51, Sep. 2018.

[91] R. J. Wieringa and M. Daneva, "Six strategies for generalizing software engineering theories," *Science of Computer Programming*, vol. 101, pp. 136–152, 2015. [Online]. Available: https://doi.org/10.1016/j.scico.2014.11.013

[92] H. Zhang and H. B. K. Tan, "An empirical study of class sizes for large java systems," in *APSEC*. IEEE Computer Society, December 2007, pp. 230–237.

[93] H. Zhang, H. B. K. Tan, and M. Marchesi, "The distribution of program sizes and its implications: An eclipse case study," in *1st International Symposium on Emerging Trends in Software Metrics*, 2009, pp. 1–10.