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 (SC20 [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 experimental 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 © 2023 IEEE
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.

![Simulink Model Diagram](Image)

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 (≤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 “inlineing” 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 “Torial”, 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 correlated1 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. SC20: 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 SC20 [7], i.e.,

1The 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 τ ranges at that level, i.e., “weak” below 0.4, then “moderate” to below 0.7, “strong” to below 0.9, etc. [34]

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 SC20’s suitability for empirical model-based research, analyzing each SC20 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 SC20 models were inadequate for most empirical research, but identified a few mature models. The work also noted that some SC20 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 SC20 (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., SC20 [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.
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.

<table>
<thead>
<tr>
<th>Corpus</th>
<th>Version of Simulink Models</th>
<th>Cut-off</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC</td>
<td>Original corpus</td>
<td>2017 [77]</td>
<td></td>
</tr>
<tr>
<td>SLNET</td>
<td>Larger corpus</td>
<td>Feb ’20 [87]</td>
<td></td>
</tr>
<tr>
<td>SC20</td>
<td>SC re-collected at later version</td>
<td>Aug ’20 [8]</td>
<td></td>
</tr>
<tr>
<td>SC_R</td>
<td>SC re-collected at SC’s version</td>
<td>2017 ×</td>
<td></td>
</tr>
<tr>
<td>SC20R</td>
<td>SC20 completed at SC20’s version</td>
<td>Aug ’20 ×</td>
<td></td>
</tr>
<tr>
<td>SC20R_Evol</td>
<td>SC20R GitHub projects’ Git histories</td>
<td>Apr ’23 ×</td>
<td></td>
</tr>
<tr>
<td>SLNET_Evol</td>
<td>SLNET GitHub projects’ Git histories</td>
<td>Apr ’23 [85]</td>
<td></td>
</tr>
</tbody>
</table>

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 straightforward 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. SC20R & SC20R_Evol Corpora to Reproduce SC20 Results

We obtained the SC20 replication package (v2) from Figshare [8], which contains 1,736 models grouped into 194 projects with non-model files removed. The SC20 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 SC20R.

To extract model metrics, SC and SC20 mostly use the Simulink API, but there are differences. For instance, SC20 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 SC20 implements McCabe’s definition (independent paths). From the SC20 paper [7] and our correspondence with the SC20 team we could not reconstruct how SC20 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 SC20 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 (“SC20R_Evol”). 3/35 repositories were no longer online.

**Finding 1:** The SC and SC20 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 SC20, 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 SC20). 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 \(^{\text{g}}\) 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\(^{\text{g}}\)) 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_R’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 (top), SC20R (middle), and SLNET (bottom); M = models; Mc = models compiling in our setup; Mh = hierarchical models; C = non-hidden connections; \( z^{\omega} \) = 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.

<table>
<thead>
<tr>
<th>Models</th>
<th>Hierarchical Blocks</th>
<th>Connections</th>
<th>Solver Step</th>
<th>Simulation Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>M</td>
<td>Mc</td>
<td>Mh</td>
<td>B</td>
</tr>
<tr>
<td>Tutorial</td>
<td>41</td>
<td>41</td>
<td>37</td>
<td>40</td>
</tr>
<tr>
<td>GitHub</td>
<td>165</td>
<td>92</td>
<td>53</td>
<td>151</td>
</tr>
<tr>
<td>M-Central</td>
<td>674</td>
<td>294</td>
<td>595</td>
<td>955</td>
</tr>
<tr>
<td>SourceForge</td>
<td>230</td>
<td>33</td>
<td>196</td>
<td>201</td>
</tr>
<tr>
<td>Other</td>
<td>7</td>
<td>4</td>
<td>3</td>
<td>7</td>
</tr>
</tbody>
</table>

| Table III: Model metric distributions per project (p) and per model (m) in SC (R), SC20R (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 |
| Models p       | 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 |
| Blocks p       | 1    | 1    | 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 |
| 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 |
| Connections p  | 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 |
| Subsystems p   | 0    | 0    | 0    | 1,294 | 1,294 | 2,117 | 9.8 | 10.0 | 12.1 | 3.0 | 3.0 | 2.0 | 44.1 | 41.8 | 75.3 |
| Cyclom. C. p   | 0    | 0    | 0    | 322  | 322  | 2,404 | 37.7 | 30.7 | 22.2 | 7.0 | 7.0 | 5.0 | 49.4 | 54.1 | 18.1 |
| Model Ref. m   | 0    | 0    | 0    | 10   | 10   | 54   | 0.0 | 0.1 | 0.1 | 0.0 | 0.0 | 0.0 | 0.4 | 0.8 | 1.5 |
| Alg. L. m      | 0    | 0    | 0    | 7    | 7    | 37   | 0.2 | 0.2 | 0.1 | 0.0 | 0.0 | 0.0 | 0.7 | 1.0 | 1.1 |
| LL Blocks m    | 0    | 0    | 0    | 657  | 423  | 2,311 | 1.9 | 1.7 | 2.0 | 0.0 | 0.0 | 0.0 | 53.8 | 48.7 | 81.1 |
| Sub. Blocks m  | 0    | 0    | 0    | 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 SC20 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-
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\textsubscript{R} (0.9%), 18 SC\textsubscript{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\textsubscript{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\textsubscript{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\textsubscript{S-fct} = models with 1+ S-functions; LQ = lower quartile; UQ = upper quartile; med = median.

<table>
<thead>
<tr>
<th></th>
<th>M\textsubscript{S-fct}</th>
<th>min</th>
<th>LQ</th>
<th>med</th>
<th>UQ</th>
<th>max</th>
<th>avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC\textsubscript{R}</td>
<td>351</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.38</td>
<td>0.92</td>
<td>0.20</td>
</tr>
<tr>
<td>SC\textsubscript{20R}</td>
<td>378</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.50</td>
<td>0.98</td>
<td>0.23</td>
</tr>
<tr>
<td>SLNET</td>
<td>1,504</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.50</td>
<td>0.99</td>
<td>0.21</td>
</tr>
</tbody>
</table>

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\textsubscript{R} models and 20% of SLNET using S-functions. For models that use S-functions, 41% of SC\textsubscript{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 \( \tau \). 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\textsubscript{R} we used non-Simple models. For SC\textsubscript{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.

<table>
<thead>
<tr>
<th></th>
<th>M</th>
<th>B</th>
<th>C</th>
<th>UB</th>
<th>MHD</th>
<th>CRB</th>
<th>NCS</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC\textsubscript{R}</td>
<td>160</td>
<td>0.29</td>
<td>0.32</td>
<td>0.31</td>
<td>0.38</td>
<td>0.28</td>
<td>0.29</td>
</tr>
<tr>
<td>SC\textsubscript{20R}</td>
<td>58</td>
<td>0.16</td>
<td>0.16</td>
<td>0.20</td>
<td>0.31</td>
<td>0.41</td>
<td>0.41</td>
</tr>
<tr>
<td>SLNET\textsubscript{200}</td>
<td>279</td>
<td>0.27</td>
<td>0.27</td>
<td>0.23</td>
<td>0.10</td>
<td>0.28</td>
<td>0.27</td>
</tr>
<tr>
<td>SLNET\textsubscript{200-300}</td>
<td>111</td>
<td>-0.02</td>
<td>0.12</td>
<td>0.16</td>
<td>0.05</td>
<td>0.07</td>
<td>0.07</td>
</tr>
</tbody>
</table>

SC\textsubscript{R} models have a weak positive correlation (0.28 to 0.38) between cyclomatic complexity and model metrics. For SC\textsubscript{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\textsubscript{Evol} and SC\textsubscript{20REvol} per-model (m) and per-project (p) change metrics; Total commits, commits per day during project duration, merge commits (\textgreater{}), and commits of 1+ mdl/slx files (MS); commit authors and commit\textsubscript{MS} authors; med = median; std = standard deviation.

<table>
<thead>
<tr>
<th>Metric</th>
<th>SC\textsubscript{20REvol}</th>
<th>SLNET\textsubscript{Evol}</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>min</td>
<td>max</td>
</tr>
<tr>
<td>Commits p</td>
<td>1</td>
<td>590</td>
</tr>
<tr>
<td>Commit / day p</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Commits\textsubscript{MS} %</td>
<td>p</td>
<td>0</td>
</tr>
<tr>
<td>Commits\textgreater{} %</td>
<td>p</td>
<td>0</td>
</tr>
<tr>
<td>Updates\textsubscript{MS} m</td>
<td>0</td>
<td>43</td>
</tr>
<tr>
<td>Authors p</td>
<td>1</td>
<td>16</td>
</tr>
<tr>
<td>Authors\textsubscript{MS} %</td>
<td>p</td>
<td>0</td>
</tr>
</tbody>
</table>

F. Converging Result: Suitability For Change Studies

To assess their applicability for Simulink model and project change studies, we analyzed SC\textsubscript{20REvol}’s 32 and SLNET\textsubscript{Evol}’s 208 git repositories (for SLNET\textsubscript{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\textsubscript{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\textsubscript{20REvol} and 15% of SLNET\textsubscript{Evol} projects have more than 50 commits. Across SC\textsubscript{20REvol} and SLNET\textsubscript{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\textsubscript{20REvol} and almost half of SLNET\textsubscript{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\textsubscript{20REvol} projects and 186 SLNET\textsubscript{Evol} projects. Even with this filtering, the figure may still be biased towards projects with fewer commits as the majority of both SC\textsubscript{20REvol} and SLNET\textsubscript{Evol} projects have less than 11 commits.

Finding 6: A quarter of SLNET\textsubscript{Evol} projects are developed collaboratively and have 1+ multi-revision models.

Implication: SLNET\textsubscript{Evol} projects have the potential to yield valuable insight into open-source Simulink development.

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\textsubscript{20}’s heuristics to search for Embedded Coder [44] or TargetLink [20] traces. But we found inconsistencies between SC\textsubscript{20}’s results (finding no code generation models) and their replication package’s heuristics [9]. During our interactions the SC\textsubscript{20} team acknowledged a bug and fixed it in their replication package version 2 [8].

Specifically, SC\textsubscript{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.

<table>
<thead>
<tr>
<th></th>
<th>EC$_{20}$</th>
<th>EC</th>
<th>GRT</th>
<th>Other</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tutorial</td>
<td>41</td>
<td>1</td>
<td>12</td>
<td>0</td>
<td>13</td>
</tr>
<tr>
<td>GitHub</td>
<td>165</td>
<td>0</td>
<td>4</td>
<td>52</td>
<td>60</td>
</tr>
<tr>
<td>MATC</td>
<td>674</td>
<td>0</td>
<td>47</td>
<td>101</td>
<td>257</td>
</tr>
<tr>
<td>Sourceforge</td>
<td>230</td>
<td>0</td>
<td>0</td>
<td>96</td>
<td>183</td>
</tr>
<tr>
<td>Others</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Σ SC$_{R}$</td>
<td>1,117</td>
<td>1</td>
<td>52</td>
<td>262</td>
<td>514</td>
</tr>
<tr>
<td>GitHub</td>
<td>1,637</td>
<td>14</td>
<td>129</td>
<td>502</td>
<td>865</td>
</tr>
<tr>
<td>MATC</td>
<td>6,239</td>
<td>19</td>
<td>423</td>
<td>1,050</td>
<td>2,706</td>
</tr>
<tr>
<td>Σ SLNET</td>
<td>7,876</td>
<td>33</td>
<td>552</td>
<td>1,552</td>
<td>2,635</td>
</tr>
</tbody>
</table>

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.