








# EvoSL: A Large Open-Source Corpus of Changes in Simulink Models & Projects

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

Alexander Boll   
*Software Engineering Group*  
*University of Bern*  
3012 Bern, Switzerland  
alexander.boll@inf.unibe.ch

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

Timo Kehrer   
*Software Engineering Group*  
*University of Bern*  
3012 Bern, Switzerland  
timo.kehrer@inf.unibe.ch

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

**Abstract**—Having readily available corpora is crucial for performing replication, reproduction, extension, and verification studies of existing research tools and techniques. MATLAB/Simulink is a de-facto standard tool in several safety-critical industries for system modeling and analysis, compiling models to code, and deploying code to embedded hardware. There is no commonly used corpus for large-scale model change studies because there is no readily available corpus. EvoSL is the first large corpus of Simulink projects that includes model and project changes and allows redistribution. EvoSL is available under a permissive open-source license and contains its collection and analysis tools. Using a subset of EvoSL, we replicated a case study of model changes on a single closed-source industrial project.

**Index Terms**—reproducibility, replication, Simulink, open science, Simulink model changes, corpus, evolution

## I. INTRODUCTION

There is currently no well-packaged, single source of open-source MATLAB/Simulink projects suitable for studying changes in Simulink models or projects. This is primarily due to the overhead associated with mining open-source repositories for such projects. For instance, GitHub's API does not readily facilitate filtering Simulink projects. Additionally, many open-source projects have been rendered inactive, adding another layer of complexity to the task of filtering out unwanted noise [27]. So creating a centralized and diverse set of Simulink projects is currently challenging.

This is a significant problem, as Simulink is a powerful tool that is widely used in several safety-critical industries such as automotive, aerospace, healthcare, and industrial automation for system modeling and analysis, compiling models to code, and deploying code to embedded hardware. As models become increasingly complex, understanding the impact of changes on the overall system becomes challenging and maintaining the consistency between models becomes harder. To alleviate the problem, researchers often collaborate with industry to study

evolution patterns and develop tools and techniques [52], [24], [39]. But this has significantly hampered the advancement of the research as the software artifacts they used are generally not made available due to confidentiality agreements hindering replication, reproduction, extension, and verification of results.

In software engineering research, there has been steady progress towards making research artifacts publicly available, which in turn has increased the impact of the research [17]. The full adoption of the open-source mindset in model-based development research has been limited, due to a prevailing view that publicly accessible models have limited research utility or because of non-disclosure agreements between researchers and industry partners [6]. Recent work has created increasingly larger corpora of open-source Simulink models [7], [9], [48], [47]. A recent empirical study has shed light on the potential of using open-source Simulink models in model evolution studies [5]. However to date these corpora do not contain Simulink model or project change data.

To address the issue, we present EvoSL, a curated corpus of 924 Git repositories consisting of over 140k commits. EvoSL is self-contained and redistributable, automatically (apart from occasional license reviews) collected from GitHub. We demonstrate EvoSL's usefulness by replicating on a EvoSL subset a model evolution study originally performed on an industry project. Our results share several similarities, while also bringing to light significant differences. For instance, our analysis found that engineers spend a substantial amount of time managing signal data rather than implementing algorithms and documentation is often neglected. To summarize, the paper makes the following major contributions.

- We created EvoSL, a corpus of 924 Simulink repositories. We mined GitHub to extract and filter Simulink-based Git repositories that permit redistribution. To the best of our knowledge, EvoSL is the first corpus of third-party

Simulink projects to perform model change studies.

- To assess EvoSL’s usefulness, we tried to replicate a prior study that analyzed changes of closed-source industrial models. We found several of the original findings could be observed on the open-source models.
- All artifacts of the paper including tools and mined data are open sourced on Zenodo [43], [44] and Figshare [45].

## II. BACKGROUND

Simulink [30] is a popular model-based development tool that allows scientists and engineers to design, analyze, and implement complex systems. For example, it is widely used in the aerospace, automotive, healthcare, and robotics industries.

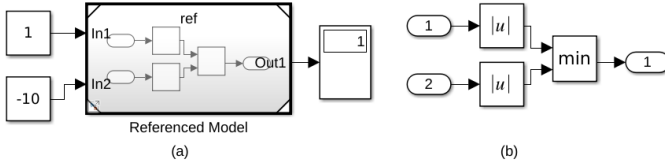


Fig. 1: Two tiny example Simulink models: The left model (a) reuses the functionality of the referenced model (b).

A Simulink user designs a system via a graphical modeling environment as a *block diagram*, by connecting parameterized blocks that represent components, signals, and mathematical operations. Figure 1 shows a tiny example. Each block processes the input it receives via *input ports* and passes its outputs via *output ports* via *signal lines* to subsequent blocks.

Simulink users can pick blocks from a wide variety of libraries and create custom blocks. A block *mask* [31] can add user-defined constraints and user-interface elements to a block. A Simulink user can annotate a model using *annotations* and alter its behavior via *configurations* [32].

Simulink’s built-in *Model Comparison Tool* [36] compares two model versions at various levels of granularity—from blocks to the overall model structure. The example in Figure 2 shows the differences between two versions of a Simulink tutorial model (sf car [33]), i.e., the addition of an Output block (Out1) and the corresponding update of the Vehicle-to-transmission connection line (partially highlighted in yellow).

### A. Studies of Changes in Simulink Models & Projects

Despite the importance of Simulink in practice, to the best of our knowledge there are only a few studies of how practitioners develop Simulink projects and how Simulink models change during development [51], [5], [22]. While studying the development history of large Simulink projects in both closed-source industrial [51], [22] and open-source development [51], such work has mostly consisted of case studies of one or two projects.

The one exception we are aware of is the recent study [5] that collected the commit histories of 35 open-source Simulink projects of an earlier corpus [9] that were still online. Besides the limited number of projects, the study also remained focused on high-level project history change data and thus did

not analyze changes within a model file, e.g., which model elements are changed and how the elements are changed.

### B. State of Open-source Simulink Corpora

While several services provide open-source Simulink models, to the best of our knowledge, none of these services can currently be used directly as a corpus of Simulink project repositories. For example, GitHub does not list Simulink as a separate language, has many projects that are unclearly licensed or are duplicates, and over time many projects disappear. Similarly, Software Heritage is not set up for frequent public download of full repositories and MATLAB Central File Exchange does not support commit-level project histories.

The lack of a corpus of Simulink repositories has been partially addressed by recent efforts to create ever-larger corpora of open-source Simulink models [7], [9], [5], [47]. On the positive side, these corpora have been used as a training set for machine-learning based approaches [48], [46] and to evaluate a variety of novel techniques [1], [48], [14], [2], [8]. Unfortunately, these corpora do not contain model changes.

To gauge the promise of open-source projects for studying Simulink project and model changes, Boll et al. [5] studied the Git repositories of 35 GitHub projects of Chowdhury et al.’s corpus [9]. With a Simulink expert many of these 35 projects were found to not mirror industrial Simulink projects for various reasons (i.e., under 50 day project duration, single author, and few merge commits). On the positive side, the study mentions three projects as promising for—due to their low total number—case study research.

While we could add model changes to one of the above corpora (e.g., by adding Git commits from GitHub), the maximum size of such a complemented corpus would still be relatively limited. Specifically, the corpus with the by-far most potentially available project histories is SLNET with its 225 GitHub projects [47].

### C. Available Open-source Simulink Project Histories

As of March 2023, the major non-GitHub services we are aware of hosting code repositories, GitLab and SourceForge, host orders of magnitude fewer open-source Simulink repositories than GitHub. Specifically, before removing projects that are empty, forks, duplicate, or have an unclear license, a quick search for “Simulink” yields 52 SourceForge projects and one GitLab project.

While Software Heritage preserves many important open-source code repositories long-term, we do not use it for the following reasons. First, Software Heritage is not meant as a primary source, downloading Software Heritage repositories is expensive and should only be done if the primary source becomes unavailable [49]. Second, it contains fewer repositories (i.e., missing over a third of the 14k EvoSL<sup>+</sup> Simulink root repositories we located on GitHub). Finally, Software Heritage currently does not provide GitHub project data such as issues, comments, and pull requests.

Beyond centralized project hosting services, a recent study [54] found the three most-used decentralized code

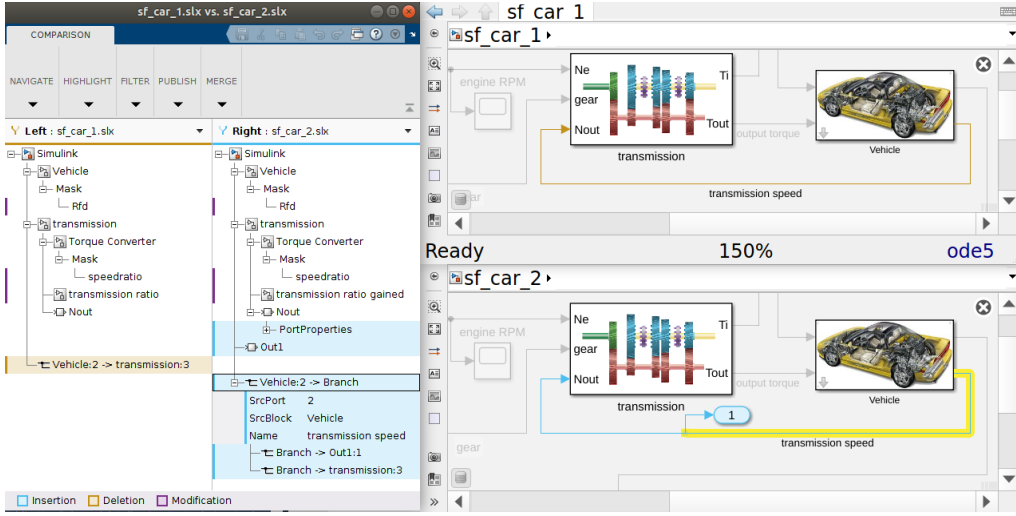


Fig. 2: Sample Simulink Model Comparison tool report comparing two versions of the sf car Simulink tutorial model [33].

repository sources GitLab Community Edition, Gogs, and Gitea to provide over 45k public open-source Git repositories, which tend to be longer-running, more academic, and more collaborative than GitHub projects. We did not attempt to mine these services as overall they had three orders of magnitude fewer projects than GitHub and do not provide uniform project data such as issues, comments, and pull requests.

### III. CORPUS OF SIMULINK MODEL & PROJECT CHANGES

Downloading Simulink projects from GitHub is not straightforward, as GitHub does not label Simulink projects. To heuristically address this issue, EvoSL-Miner queried GitHub’s public REST API (via PyGitHub [20]) in February and March 2023 to first download *root* repositories (i.e., not forks) that (a) are marked as using the MATLAB programming language or (b) match when searching their repository name, description, or README file for “Simulink”. EvoSL-Miner extends SLNET-Miner, by downloading the full Git repository instead of a snapshot, while still satisfying the GitHub REST API limits (30 search requests per minute per authenticated user, yielding 1k results per request; 5k other requests per hour per authenticated user) [12]. This yielded over 360k such MATLAB/“Simulink” projects.

In the second step (labeled “2” in Figure 3), we only keep a MATLAB / “Simulink” project in EvoSL<sup>+</sup> (and download from the GitHub API its summary data, issues, pull requests, and comments) if the project’s latest default-branch version has at least one mdl or slx file we can open with Simulink 2022b (Simulink’s default file format changed with the R2012b release from the proprietary ASCII mdl file to the (binary) zip container slx). This yielded EvoSL<sup>+</sup>’s 13,919 root Simulink Git repositories with metadata. (3) Third, we use the root project metadata to similarly download all (transitive) project forks, yielding EvoSL<sup>+</sup>’s 13,786 Simulink fork Git repositories and their metadata.

Table I summarizes the further pre-processing, which adds to EvoSL<sup>+</sup>’s metadata but for license and storage space reasons

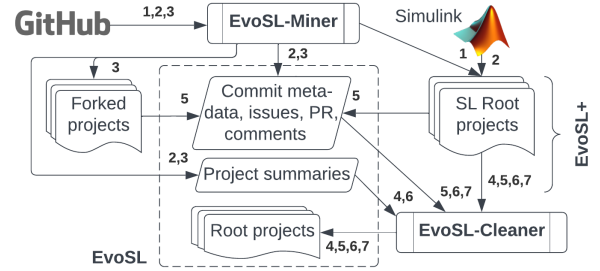


Fig. 3: Overview of EvoSL collection and cleaning steps: EvoSL-Miner downloads EvoSL<sup>+</sup> (Git projects and metadata), from which EvoSL-Cleaner removes certain Git repositories.

TABLE I: Data cleaning steps: Root = project with 1+ Simulink models; License = has a license; Permissive = license allows re-distribution; MC = has 2+ model commits; ND = no duplicates; EvoSL = has model with 2+ commits.

Root	License	Permissive	MC	ND	EvoSL
13,919	2,323	2,282	1,081	1,071	924

only includes the EvoSL subset (Git repositories and all metadata including issues, comments, pull requests, etc.) in the EvoSL distribution. (4) Fourth, EvoSL-Cleaner only includes an EvoSL<sup>+</sup> root project in EvoSL if the project has a license (2,323/13,919 projects) and the license allows redistribution. GitHub has a structured way for authors to set their project’s license, which GitHub then converts into a corresponding license file (and subsequently exposes via an API). For the 291 project licenses GitHub did not understand (i.e., the API returns “Other”), we realized on manual review that many of them just appear to be common open-source licenses applied manually—without using GitHub’s structured license settings. We conservatively judged 250/291 projects to allow

redistribution.

While for many applications (e.g., as a machine learning training set) a larger corpus is better, we also had to satisfy long term storage size limitations. We thus (5) prioritized the projects with the most changes to Simulink model files (as opposed to changes to other files). Specifically, we extract commit metadata via PyDriller [50]. While EvoSL contains the full Git repositories and all issues, pull requests, and comments, we configure PyDriller to only process the commits of each project’s default-branch. We then only keep a Git repository in EvoSL if it has at least two commits across its (default-branch) Simulink model files, yielding 1,081 Git repositories with Simulink model commits.

After removing forks, the dataset may still contain other duplicates [37], which we remove heuristically. In step (6) we first mark two EvoSL Git repositories as potential duplicates if they have the same Figure 4 Project\_Commit\_Summary metric values (e.g., the same number of default-branch commits, same number of default-branch merge commits, etc.) and confirm this if they also have the same commit hashes. We keep the Git repository with the smaller GitHub project ID, yielding 1,071 Git repositories. (7) Finally, we remove Git repositories that do not change any default-branch Simulink model after that model file’s initial (“check-in”) commit, yielding EvoSL’s 924 Git repositories.

Finally, we compare the size of EvoSL to the largest open-source Simulink corpus to date—SLNET (which does not contain project histories). To ease comparison, we focus on the 36 EvoSL projects we could open with Simulink R2019a that have the most commits of mdl/slx files (“EvoSL<sub>36</sub>”). The latest version of the main branch of the projects in this EvoSL subset alone contains 714k Simulink blocks, significantly more than all of SLNET’s 190k blocks in its 225 GitHub projects.

#### A. EvoSL Long-term Storage and Metadata

At writing we were aware of three permanent storage options that are publicly accessible, easy to cite, and offer free data deposit and download. Since Dataverse’s 2.5 GB per-file limit [16] conflicts with some EvoSL projects and Figshare’s free 20 GB per-user limit [10] restricts EvoSL’s overall size, we upload EvoSL into Zenodo’s 200 GB hard limit [43].

What EvoSL adds to the repositories (bundling, some metadata, etc.) has the permissive CC BY 4.0 license. The distribution contains EvoSL’s 924 full Simulink Git root repositories (last updated in early March 2023). The size of the distribution is some 73 GB. Each of the 924 projects is in a zipped folder named after the project’s GitHub project ID.

In addition to the 924 full Simulink Git root repositories, EvoSL also contains the Figure 4 metadata. Specifically, the metadata is in a SQLite<sup>1</sup> database. The metadata mainly records the information EvoSL-Miner downloaded from the GitHub API, e.g., project popularity and engagement, issues, pull requests, and comments associated with issues and pull

requests. To make it easier to select projects with certain Simulink model changes, we derive and add metadata.

First, for each project’s default-branch we break down every commit that changes Simulink models into one *model commit* per model changed by that (project) commit. Specifically, we process each Git repository’s default-branch commits to create one model commit per Simulink model whose slx or mdl file was touched by a given project commit (Figure 4 Model\_Commits). A commit belonging to a merge commit is distinguished by listing more than one parent commit. We further summarize commits, e.g., the total number of default-branch commits, their number of authors, and a Simulink model’s *lifetime*—i.e., the difference between a model file’s first and last default-branch commit (Figure 4 Project\_Commit\_Summary and Model\_Commit\_Summary).

TABLE II: Simulink root projects before (EvoSL<sup>+</sup>) and after filtering (EvoSL): Issues, pull requests (PR), comments on issues and pull requests, and default-branch commits.

	Projects	Commits	Issues	PR	Comments
EvoSL <sup>+</sup> Root	13,919	419,404	5,973	7,490	14,923
EvoSL (Root)	924	143,571	3,228	1,933	10,290

Finally, Table II compares the amount of metadata for EvoSL<sup>+</sup> and EvoSL. From a project change data perspective, EvoSL is clearly an interesting (but non-representative) sample of the full EvoSL<sup>+</sup> root projects. For example, while EvoSL<sup>+</sup> contains over 15 times of the root projects of EvoSL, EvoSL contains over one third of all EvoSL<sup>+</sup> default-branch project commits and over two thirds of all EvoSL<sup>+</sup> issue and pull request comments.

#### B. Overview of EvoSL’s Simulink Model and Project Changes

It is well-known that for the entirety of GitHub the distribution of commits over projects is heavily skewed toward a few very active projects, with a long tail of projects having under 50 commits [27]. It is thus no surprise that GitHub’s Simulink projects default branches follow a similar long-tail distribution (Figure 5). For example, in EvoSL<sup>+</sup> the median number of default-branch project commits is six and the median number of model commits per project default-branch is one. Further, 91% of projects have under 50 total default-branch commits and 56% of projects only have one model default-branch commit.

To better understand model change timing and the share of Simulink models that is changed during the project, Figure 6 breaks each project default-branch’s duration into 10 buckets of equal length (normalized to each project default-branch’s duration). Here project duration is the duration from a project default-branch’s first to last commit as recorded by the timestamps assigned by the committers’ 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) [11].

<sup>1</sup>SQLite is widely used, free, self-contained, server-less, zero-configuration, backwards compatible, and cross-platform.

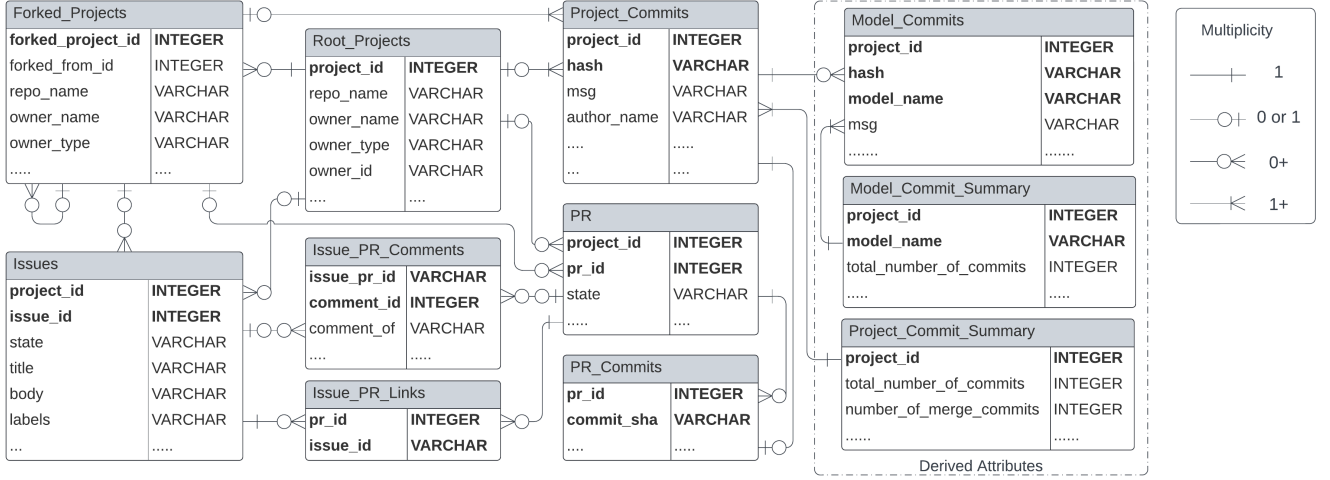


Fig. 4: EvoSL’s metadata relational database schema with multiplicity constraints, e.g.: each (default-branch) project commit is broken down into one model commit per Simulink model file change and each model commit is part of one project commit; bold = primary key; forked projects do not contain their parent project’s commits (except for one initial commit).

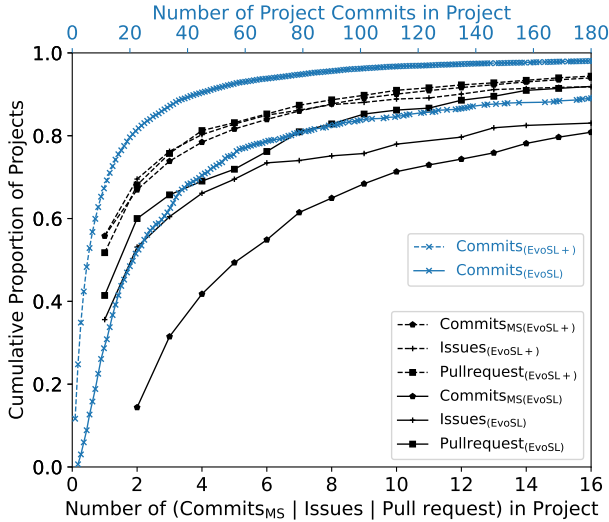


Fig. 5: Root project percentage (y-axis) with up to the given number of default-branch commits, default-branch commits of 1+ mdl/slx files, issues, and pull requests (x-axes).

Specifically, if a project only has one default-branch commit then Figure 6 assigns this commit (only) to the last bucket (90–100%). This explains Figure 6a’s spike in the 90–100% bucket, as EvoSL<sup>+</sup> is skewed towards projects with few default-branch commits. Similarly, the spike in its 0–10% bucket says that many EvoSL<sup>+</sup> projects commit (or “dump”) much of their changes (and especially changes to Simulink model files) together around the time of the initial default-branch commit.

The last-bucket concentration of commits, commits of 1+ mdl/slx files (commits<sub>MS</sub>), and share of mdl/slx files in a commit (“models under development”) all decrease in EvoSL, again when looking at the subset of EvoSL projects with 10+

default-branch commits<sub>MS</sub> (Figure 6c), and yet again when looking at the 36 top-commits<sub>MS</sub> EvoSL projects we could open with Simulink R2019a (Figure 6d). At the same time, these measures trend upward for the non-start/finish buckets. For example, for EvoSL projects with 10+ default-branch commits<sub>MS</sub>, each bucket contains commits that overall include at least 10% of the default-branch mdl/slx files.

The Figure 6 distributions over normalized project lengths further resemble traditional software development projects (rather than file dumps) when putting them into context of the Table III project metrics. First, EvoSL’s, EvoSL<sup>+</sup>’s, and EvoSL<sub>36</sub>’s absolute project lengths range up to 16 years, with average project lengths of 116, 443, and 1,553 days and median project lengths of 4, 135, and 1,207 days. Finally, EvoSL and EvoSL<sub>36</sub> projects have a median of two and 11 authors, of which more than half also commit slx/mdl files.

#### IV. REPLICATING AN INDUSTRIAL STUDY WITH EVO SL

Our main contribution is the EvoSL corpus itself, as it allows exploring various research questions on Simulink model and project changes, including commits, GitHub issues, pull requests, other project metadata, and their correlations. While open-source projects will never be exactly like industrial closed-source development in all aspects, **here we are asking if studying open-source projects can yield results that are comparable to studying closed-source Simulink projects.**

At a minimum, this would allow the research community to develop hypotheses and tools that could then be tested and validated more easily in an academic-industrial collaboration. This may lead to faster progress than relying on all the heavy lifting of developing hypotheses and tooling from scratch being done in closed-source academic-industrial collaborations.

To explore this question, we pick one recent representative empirical study of the changes in a closed-source Simulink project and replicate the study using EvoSL projects. Jaskolka



TABLE III: Default-branch metrics: Commits, commits per day during project duration, merge commits ( $\succ$ ), and commits of 1+ mdl/slx files (MS); commit authors and commit<sub>MS</sub> authors; l = low; h = high; med = median; std = standard deviation.

Default-branch:	EvoSL <sup>+</sup> (root)					EvoSL (root)					EvoSL <sub>36</sub> (root)				
	l	h	avg	med	std	l	h	avg	med	std	l	h	avg	med	std
Commits	1	15,060	30	6	305	2	15,060	155	22	992	102	1,452	499	381	349
Commits / day	0	91	2	1	4	0	50	1	0	3	0	1	0	0	0
Commits <sub>&gt;</sub> [%]	0	55	2	0	6	0	48	5	0	7	0	17	9	9	5
Commits <sub>MS</sub> [%]	0	100	43	33	31	0	100	36	31	25	6	88	35	27	19
Authors	1	103	2	1	3	1	103	4	2	8	1	45	14	11	11
Authors <sub>MS</sub> [%]	1	100	84	100	25	2	100	73	67	28	25	100	62	56	21
Durations [days]	0	5,909	116	4	326	0	5,909	443	135	703	264	5,909	1,553	1,207	1,076

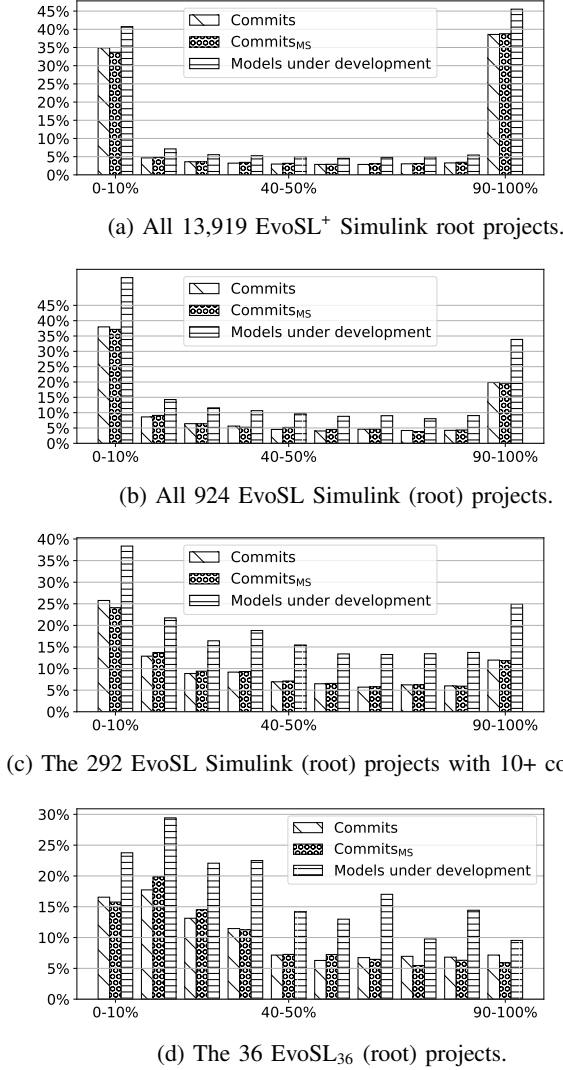


Fig. 6: Across projects' normalized duration on x-axis: Total default-branch commits, default-branch commits<sub>MS</sub>, and percentage of mdl/slx files included in bucket's commits.

et al. [22] examined changes across different model versions of a proprietary industrial software repository of an automotive control system to understand how Simulink models evolve over time. Their analysis shows that well-accepted software

engineering principles (such as low degree of change to interfaces) are not practiced and engineers spend significant amounts of time in non-value added work such as migrating the project to new Simulink versions. Specifically, we are investigating the following three research questions, all copied verbatim from Jaskolka et al. [22].

RQ1 What basic elements change the most?

RQ2 Which blocks are involved in changes most frequently?

RQ3 Which are identified categories of change?

#### A. Experimental Setup Following C-study

To replicate the earlier study [22] (which we call C-study, where C may stand for closed-source or car industry), we try to follow C-study's setup and procedures as closely as possible, using both the same conceptual framework of Simulink model changes and tooling for collecting change data.

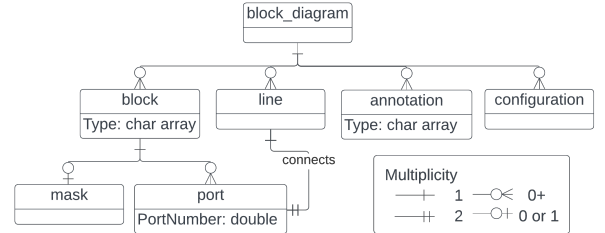


Fig. 7: Simplified Simulink meta-model: 6 element types [22].

For the conceptual framework, Figure 7 summarizes the relevant parts of C-study's meta-model of Simulink models. Here a block diagram is composed of six *element* types—block, line, port, mask, annotation, and configuration. For each of these six element types, C-study collects four change types—add, delete, rename, and (otherwise) modify.

C-study detects a change by comparing two snapshots of a model. An *added* element does not exist in the before- but in the after-model. A *deleted* element exists in the before- but not in the after-model. A *renamed* and *modified* element each exist in both the before- and the after-model but have their name or another parameter changed.

On the tooling side, Figure 8 gives an overview of how we adapted C-study's model change computation. C-study queried a Rational Synergy commercial issue tracking system

to extract before- and after- model file versions from a Rational Change commercial change management system [18], [19]. Given EvoSL’s use of standard Git repositories and its inferred model commit metadata, it is straight-forward to similarly extract such before- and after- model file versions from EvoSL.

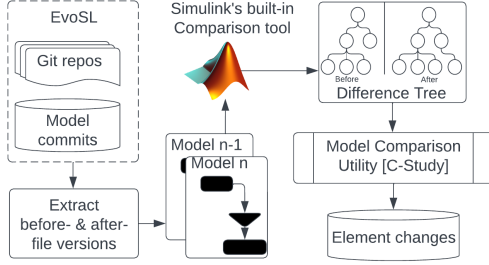


Fig. 8: Using C-study’s Model Comparison Utility [22], [21] to mine Simulink model changes in EvoSL.

Since we do not know how C-study treated merge commits (or commits from non-default branches) we focus on non-merge default-branch commits. C-study’s open-source Model Comparison Utility [22], [21] passes each pair of before- and after- file versions to Simulink’s built-in model comparison tool and breaks its output down into individual model element changes. Due to Simulink API limitations, C-study’s Model Comparison Utility discards non-functional changes (such as layout) and block defaults. We store and distribute the remaining derived element changes in a SQLite database.

For this replication study we had to make a trade-off between model selection and replication accuracy. The key reason is that using two different Simulink versions and their built-in model comparison tools on the same before- and after-model pair can produce vastly different results, even when both Simulink versions directly support the model file versions.

Concretely, as C-study used Simulink R2019a we passed random Simulink models developed with R2019a to both Simulink R2019a and the (ostensibly backward-compatible) R2022b. Despite the tool documentation being silent on this, about a quarter of changes were only reported by one of the two Simulink versions, with no report being a superset of the other. As we cannot run R2022b on the closed-source C-study repository we are stuck with Simulink R2019a and (as Simulink is not directly forward-compatible) we exclude from this study model files developed with R2019b or later that Simulink R2019a refuses to open.

### B. Simulink Model Changes From EvoSL Sample: EvoSL<sub>36</sub>

As C-study focuses on Simulink model changes we pick the 50 EvoSL projects that have the most default-branch changes to mdl/slx files (“commits<sub>MS</sub>”). Due to our experimental setup constraints, we remove 9 (newer) projects Simulink R2019a cannot open. We remove five additional projects who have a subset of the default-branch commits<sub>MS</sub> of another project. (EvoSL removed all explicit fork projects and exact

non-fork duplicate project histories but not such non-forked almost-duplicates.) We are thus left with 36 EvoSL projects (“EvoSL<sub>36</sub>”). Following C-study, we ran its Model Comparison Utility [22], [21] on each default-branch before- and after-commit model pair of EvoSL<sub>36</sub>, yielding 590,300 Simulink model changes (C-study analyzed 2.8M such changes).

TABLE IV: Basic project metrics copied from C-study [22] and EvoSL<sub>36</sub>’s default-branch distributions; l = low; h = high; med = median; std = standard deviation; m = model size.

	[22]	EvoSL <sub>36</sub> (default branches)				
		l	h	avg	med	std
Largest m. [blocks]	37,814	6	51,655	2,368	357	8,642
Avg m. [blocks]	1,200	5	8,366	424	113	1,438
Commits <sub>MS</sub>	1,354	60	598	141	111	105
Duration [months]	75	9	197	52	40	36
Changed model files	3,945	1	176	25	13	37

Table IV puts EvoSL<sub>36</sub> in context of C-study’s published basic project characteristics [22]. While EvoSL<sub>36</sub> has default-branch changes in fewer mdl/slx files (some 900 in total vs 3,945), has fewer total default-branch commits<sub>MS</sub> in a single project (598 vs 1,354), and has a lower average default-branch model size (424 vs 1,200 blocks), on all the measures, except changed model files, EvoSL<sub>36</sub> is within the same order of magnitude as C-study. On the flip side, EvoSL<sub>36</sub> has a larger maximum default-branch model size (52k vs. 38k blocks), longer maximum default-branch project duration (16 vs. 6 years), and more total default-branch commits with mdl/slx file changes in total (5k vs 1,354 commits<sub>MS</sub>)—again all within the same order of magnitude.

To further gauge EvoSL<sub>36</sub>’s suitability, we checked the criteria recently laid out with the help of a Simulink expert [5] when analyzing the suitability of an earlier corpus [9]. Of these criteria, we did not reach a conclusion on a steady increase of commits towards project end (as we do not know EvoSL<sub>36</sub> projects’ future timelines) and a non-low percentage of commits being merge commits (it is unclear if EvoSL<sub>36</sub>’s median 9% of commits meets this bar).

That paper’s remaining provided metrics and example values [5] all largely align with EvoSL<sub>36</sub> (Table III), i.e., a high project duration (2.3k vs. EvoSL<sub>36</sub>’s median 1.2k days in the default-branch), many authors (16 vs. median 11), many project commits (589 vs. median 381), and many default-branch commits affecting mds/slx files (44% vs. median 27%). Together with our manual sampling of commit messages, we conclude that EvoSL<sub>36</sub> projects are not synthetic outliers generated by a random generator, but represent suitable human development activity.

### C. RQ1: What Basic Elements Change the Most?

Breaking EvoSL<sub>36</sub>’s 590,300 default-branch element changes down by element and change type as C-study did yields Table V’s right columns. To ease comparison, we normalize each element type’s change count (e.g., EvoSL<sub>36</sub>’s 30k block renames) by dividing that element type’s total

changes (e.g., EvoSL<sub>36</sub>'s 300k total block changes—yielding EvoSL<sub>36</sub>'s 10.1% block rename rate).

C-study makes several observations about this element change breakdown and draws two main conclusions, all of which could equally be done with EvoSL<sub>36</sub>'s corresponding (default-branch) data, as follows. (1) First, C-study observes that the most frequently changed element type is blocks, which aligns with 300k of 590k total EvoSL<sub>36</sub> changes being blocks. (2) Second, C-study finds that line changes follow closely behind block changes, which again aligns closely with EvoSL<sub>36</sub>'s 246k line vs. 300k block changes.

(3) Third, C-study notices that line changes are dominated by first add (59%) and then delete (39%), which similarly occurs in EvoSL<sub>36</sub> (55% add and 43% delete). Combined with add- and delete-line having higher absolute numbers than add- and delete-block, C-study explains how replacing a block triggers a deleting and adding a line.

(4) Fourth, not directly referencing any additional data, C-study determines that a similar dynamic is at play with ports. As all of the previous data observations are equally true in EvoSL<sub>36</sub>, we could have equally used EvoSL<sub>36</sub> to conclude to omit both line and port changes from further analysis. (5) Finally, C-study observes that masks, annotations, and configurations have the least changes (some 3% overall), which again aligns with some 3% in EvoSL<sub>36</sub>.

Beyond C-study's observations, there are several other similarities. For example, in both data sets the most common change type is add, followed by (in order) delete, modify, and rename. The one big outlier in both datasets is configuration, which is dominated by modify and followed by add. In both datasets blocks have the highest rename rate, followed by annotations, and lower rates for the remaining element types.

*Insight 1: Besides additional similarities, all observations C-study makes about its change data are equally true for EvoSL<sub>36</sub>. We thus assume researchers could draw the same conclusions C-study drew.*

#### D. RQ2: Most Frequently-changed Block Types

To analyze which blocks change most frequently, C-study aggregates blocks by their block type (as given by the block's BlockType parameter). C-study's five block types with the most changed block instances are Inport (11.8% of all C-study changes), Outport (9.2%), From (5.4%), Constant (4.4%), and SubSystem (4.2%). These five block types are also in EvoSL<sub>36</sub>'s top-six block types by most block instance changes. The one exception is the Reference block type, which dominates EvoSL<sub>36</sub> likely because it represents custom blocks we did not load from one of EvoSL<sub>36</sub>'s custom libraries.

Figure 9 shows all block types whose block instances have over 50 changes across the EvoSL<sub>36</sub> default branches. The frequency of the six block types with the most block instance changes are Reference (16% of all EvoSL<sub>36</sub> changes), SubSystem (7.3%), Inport (4.9%), Outport (3.5%), Constant (1.8%), and From (1.5%).

Besides Reference, EvoSL<sub>36</sub> also has a higher rate of Subsystem changes. Subsystem blocks are typically used to modularize and organize a large model into smaller and more manageable components. We assume C-study's slightly lower rate of Subsystem block changes stems from each project tending to become relatively more stable over time. This progression playing out for each EvoSL<sub>36</sub> project would explain EvoSL<sub>36</sub>'s overall higher rate of such structural changes. EvoSL<sub>36</sub>'s next two most frequently changed block types, Inport and Outport, appear in this order also in C-study.

Finally, we examine the ordering of block types by most-changed total block instances with Kendall's rank and p-value less than 0.05. Based on this test, the trend of most-to-least frequently changed block types in C-study and EvoSL<sub>36</sub> are strongly positively correlated ( $\tau = 0.99$ ).

*Insight 2: EvoSL<sub>36</sub> mimics several characteristics of C-study's block type change distribution, including the most-changed block types and a strong correlation between the order of block types by block changes.*

#### E. RQ3. Which Are Identified Categories of Change?

The standard Simulink language block libraries categorize blocks pertaining to their purpose or a common quality. However, block types in these groups overlap with the other groups. To categorize each block type to a non-overlapping category, Jaskolka et al. created a new category scheme in which each block type falls under a single category according to their purpose. They also introduced new categories such as Documentation and Interface. Table VI shows the list of categories with some example blocks, and full details can be found in their work [23]. We adopted the new category scheme and analyzed the changes according to it. Block types not listed in Jaskolka's category scheme we marked as "Others".

Table VII shows the ratio of (default-branch) block changes by block category. Most EvoSL<sub>36</sub> changes are on structural blocks, followed by signal routing, math, and source blocks. Unlike C-study where interface changes contributed to over one-third of block changes, in EvoSL<sub>36</sub> interfaces are stable with under 1% of block changes, indicating good modeling practices. We delve into a few categories below.

##### a) Changes to Signal Routing and Structural Blocks:

In Simulink, data produced and processed by blocks is routed via signal lines. Rather than analyzing the signal lines (as discussed in Section IV-C), analyzing changes to the blocks that are responsible for routing, combining, creating, and selecting data is more revealing. Table VII shows that engineers spend a substantial amount of time managing signal data, as 29% of block changes are signal routing changes.

The many changes to signal routing blocks go hand in hand with the most frequently changed block category, i.e., Structural. In model based development, complexity is abstracted through creating hierarchical models. The many changes to SubSystem and Reference blocks (Figure 9), contributed to significant changes to signal routing between the models' hierarchical layers.



TABLE V: Types of Simulink model element changes in C-study and EvoSL<sub>36</sub> (default branches); normalized = element type’s specific changes divided by that element type’s total changes; Re = rename; Mod = modify; Del = delete; EC/TC = element type’s total changes divided by total changes; Anno = annotation; Conf = configuration.

	C-study (normalized)					EvoSL <sub>36</sub> (normalized)					EvoSL <sub>36</sub> (absolute)				
	Re	Mod	Del	Add	EC/TC	Re	Mod	Del	Add	EC/TC	Re	Mod	Del	Add	Total
Block	12.0	24.9	22.9	40.2	55.3	10.1	35.4	23.4	31.1	50.7	30,183	106,093	69,985	93,295	299,556
Line	0.2	2.0	39.2	58.6	38.9	0.2	1.8	43.3	54.6	41.6	558	4,385	106,575	134,340	245,858
Port	0.3	27.6	27.6	44.5	3.2	0.0	0.5	43.4	56.1	4.2	0	124	10,774	13,951	24,849
Mask	0.0	19.8	16.9	63.2	1.8	0.0	43.7	23.7	32.5	1.2	0	3,148	1,709	2,343	7,200
Anno	4.0	10.4	34.2	51.4	0.8	6.3	7.4	44.3	42.0	1.1	407	482	2,871	2,723	6,483
Conf	0.0	98.5	0.0	1.5	0.0	2.2	65.0	3.5	29.3	1.1	142	4,130	220	1,862	6,354
All	6.7	15.9	29.4	48.0	100.0	5.3	20.1	32.5	42.1	100.0	31,290	118,362	192,134	248,514	590,300

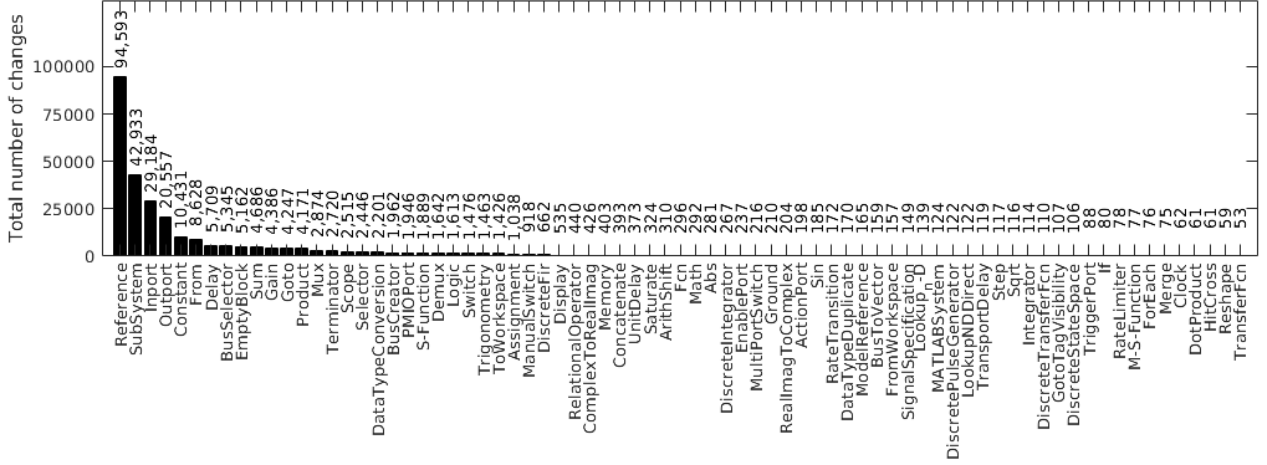


Fig. 9: EvoSL<sub>36</sub>’s 299,556 (default-branch) Simulink block changes by block type (showing block types with 50+ changes).

TABLE VI: C-study’s 13 Simulink block categories [22].

Category	Example Blocks
(Model) Interface	root-level Inport/Outport, global DataStoreRead/DataStoreWrite, FromFile/ToFile, FromSpreadsheet, ToWorkspace/FromWorkspace
Signal Routing	non-root Inport/Outport, Goto/From, local DataStoreRead/Write, BusCreator, Merge, Assignment
Signal Attributes	RateTransition, DataTypeDuplicate, Signal- Conversion, DataTypeConversion
Structural	SubSystem, Reference
Conditional	If, Switch, SwitchCase, ManualSwitch
Discrete	Delay, UnitDelay, Filter, Integrator
Math	Sum, MinMax, Rounding, Abs, Gain
Logic	RelationalOperator, LogicalOperator
Trigger	TriggerPort, EnablePort, ActionPort
Sources	Ground, Step, Clock, Constant
Sinks	Terminator, Scope, Display
Documentation	ModelInfo, DocBlock
Custom	S-Function

b) *Changes to Documentation Blocks*: Simulink provides various options for documenting models. Similar to code comments for understanding textual programs, in Simulink one can use annotations including text and images embedded in the model. Users can also embed plain text or a document via a

TABLE VII: EvoSL<sub>36</sub>’s block changes by C-study’s Table VI categories; Others = all newer and uncategorised blocks types.

Block Category	%	Block Category	%
Structural	49.55	Signal Routing	28.90
Math	6.18	Sources	3.96
Others	3.10	Discrete	2.80
Sinks	2.08	Signal Attributes	1.03
Conditional	0.97	Logic	0.85
Custom	0.75	Interface	0.62
Trigger	0.19	Documentation	0.01

*DocBlock* [34], which may contain a more thorough description of the design. Finally, Simulink’s *Model Info* [35] block shows (automatically updating) revision control information such as creator and the last modified date.

Compared to block changes overall, EvoSL<sub>36</sub> has significantly fewer changes to documentation-related blocks. The low documentation change frequency is inline with existing software documentation for Simulink models [38] and most other software systems [26], [3]. 99% of the documentation-related changes are made through annotations. Unlike in industrial development where Model Info is encouraged to keep track of the original model’s creator and other metadata, the EvoSL<sub>36</sub> open-source projects do not follow the practice.

*Insight 3: As in C-study, EvoSL<sub>36</sub> engineers made much more changes to signal routing and structural blocks than to implementing algorithms, leading to the same key conclusion as C-study.*

## V. THREATS TO VALIDITY

One threat to validity is that we do not know if C-study analyzed model changes from production and all development branches. As C-study does not mention branches at all [22], [23], our replication focuses on model changes that are either direct commits to the default branch or added to the default branch via merging. By skipping analysis of commits from branches not ultimately merged back into the default branch (including those that got removed via squashing before merging to the default branch), we missed 1,084 commits<sub>MS</sub> (and their 2,540 model commits), beyond the 5,070 commits<sub>MS</sub> (and their 9,845 model commits) we analyzed for EvoSL<sub>36</sub>.

EvoSL is curated from GitHub and may not be representative of all open-source Simulink repositories. GitHub does not recognize Simulink as a programming language. We did a thorough search via GitHub’s API, filtering GitHub projects written in MATLAB on top of a “Simulink” keyword search. Note that projects that only contain Simulink models are not tagged with a programming language, so our search may have missed them. It is impractical to search 330 million GitHub repositories to filter Simulink projects.

C-study’s Model Comparison Utility captures Stateflow block changes in the model snapshots but did not label any of the EvoSL<sub>36</sub>’s change with Stateflow. To assess EvoSL’s potential to facilitate research on Stateflow changes, we ran the utility using MATLAB/Simulink 2022b on EvoSL, which yielded no Stateflow-related changes. Our attempts to identify relevant projects were also unsuccessful, despite finding at least 15 projects that mentioned Stateflow in their descriptions in our metadata. While EvoSL may not be suitable for Stateflow change studies, a few EvoSL projects could still contain Stateflow-related blocks. Also, we provide the EvoSL element change data with its metadata for further analysis.

## VI. RELATED WORK

Simulink models have largely been curated with manual or semi-automated approaches [7], [5], [41], [9]. Sánchez et al. used Google’s BigQuery to filter and extract the largest open-source Simulink models to test their tool [41]. Chowdhury et al.’s SLforge project performed the first large-scale study of 391 Simulink models, whose primary focus was to test the Simulink tool chain [7]. The authors later extended their corpus to 1071 publicly available models [9]. Boll et al. reproduced the corpus developed by Chowdhury et al., providing deeper insights on the models and modeling characteristics [5]. Shrestha et al. pioneered fully automated mining of some 400 non-hierarchical models to train a deep learning model for Simulink tool chain testing and later curated the first self-contained corpus of Simulink models, addressing limitation of earlier corpora [48], [47]. Unlike EvoSL, none of these corpora offer projects with full revision history.

Existing work in the field of model evolution is focused on clone detection, variant management, and studying the synchronous co-evolution of models and tests. To detect clones, Stephen et al. developed the SIMONE algorithm, which has been used to track the evolution of model clones and refactor cloned fragments into a library [52], [24]. Haber et al. proposed delta operations, including add, remove, modify and replace elements, to obtain desired variant models [15]. Schlie et al. focused on improving variability mining approaches by adding blocks and hierarchical levels [42]. Rapos et al. employed Simulink’s built-in comparison tool to extract change information and investigate the synchronous co-evolution of models and tests in closed-source industrial models [39]. Jaskolks et al.’s case study stands out for its comprehensive classification of changes to model elements, which we replicated in this study using EvoSL.

Mining source code from software repositories has yielded rich information researchers have leveraged for code-based research [25], [53], [4]. GitHub especially has emerged as a primary source of open-source repositories for empirical research. Consequently, researchers have developed several tools to facilitate mining from GitHub [40], [13], [50], [20]. In this study, we used PyDriller and PyGitHub to curate EvoSL [50], [20]. In recent years, there has been increasing interest in mining model-based artifacts. The MAR search engine [28], [29] has been developed to facilitate model-driven engineering efforts, i.e., the tool searches existing corpora for Simulink models. On the other hand, tools such as ModelMine allow for artifact searches directly from GitHub by searching based on file extensions. However, the tool incorrectly labels the Simulink model file extension as “.simulink”.

## VII. CONCLUSIONS

In this study, we emphasize the importance of readily accessible corpora for performing replication, reproduction, extension, and verification studies. Several safety-critical industries use MATLAB/Simulink as a standard tool for system modeling and analysis, necessitating large-scale model evolution studies. However, there has been no readily accessible corpus for such studies. To address this gap, we introduced EvoSL as the first large corpus of Simulink projects, including model and project changes, which is available under a permissive open-source license and included its collection and analysis tools. On a EvoSL subset we successfully replicated a case study of model changes in a closed-source industrial project.

## 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 *Proc. ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, 2021, pp. 427–436.
- [2] —, “SimIMA: A virtual Simulink intelligent modeling assistant,” *Software and Systems Modeling*, pp. 1619–1374, 2023. [Online]. Available: <https://doi.org/10.1007/s10270-023-01093-6>
- [3] E. Aghajani, C. Nagy, O. L. Vega-Márquez, M. Linares-Vásquez, L. Moreno, G. Bavota, and M. Lanza, “Software documentation issues unveiled,” in *Proc. 41st IEEE/ACM International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 1199–1210.
- [4] M. Allamanis and C. Sutton, “Mining source code repositories at massive scale using language modeling,” in *Proc. 10th Working Conference on Mining Software Repositories (MSR)*. IEEE, May 2013, pp. 207–216. [Online]. Available: <https://doi.org/10.1109/MSR.2013.6624029>
- [5] 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. 1–20, 2021.
- [6] 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, 2022.
- [7] 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 *Proc. 40th ACM/IEEE International Conference on Software Engineering (ICSE)*. ACM, May 2018, pp. 981–992.
- [8] 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.
- [9] 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 *Proc. 4th International Workshop on Software Engineering for Smart Cyber-Physical Systems (SEsCPS)*. ACM, May 2018, pp. 45–48.
- [10] Digital Science, “Figshare,” 2023, april 2023. [Online]. Available: <https://figshare.com/>
- [11] S. W. Flint, J. Chauhan, and R. Dyer, “Pitfalls and guidelines for using time-based Git data,” *Empirical Software Engineering*, vol. 27, no. 7, pp. 1–55, Dec. 2022.
- [12] GitHub Inc, “Rate limits,” 2023, accessed March 2023. [Online]. Available: <https://docs.github.com/en/rest/overview/resources-in-the-rest-api?apiVersion=2022-11-28#rate-limiting>
- [13] G. Gousios and D. Spinellis, “GHTorrent: Github’s data from a firehose,” in *Proc. 9th IEEE Working Conference of Mining Software Repositories (MSR)*. IEEE, Jun. 2012, pp. 12–21.
- [14] S. Guo, H. Jiang, Z. Xu, X. Li, Z. Ren, Z. Zhou, and R. Chen, “Detecting Simulink compiler bugs via controllable zombie blocks mutation,” in *Proc. 30th ACM Symposium on the Foundations of Software Engineering (FSE)*. ACM, Nov. 2022, pp. 1061–1072.
- [15] A. Haber, C. Kolassa, P. Manhart, P. M. S. Nazari, B. Rumpe, and I. Schaefer, “First-class variability modeling in matlab/simulink,” in *Proc. 7th International Workshop on Variability Modelling of Software-intensive Systems (VaMoS)*. ACM, Jan. 2013, pp. 4:1–4:8. [Online]. Available: <https://doi.org/10.1145/2430502.2430508>
- [16] Harvard Dataverse Project, “Harvard dataverse,” 2023, april 2023. [Online]. Available: <https://dataverse.harvard.edu/>
- [17] R. Heumüller, S. Nielebock, J. Krüger, and F. Ortmeier, “Publish or perish, but do not forget your software artifacts,” *Empir. Softw. Eng.*, vol. 25, no. 6, pp. 4585–4616, 2020. [Online]. Available: <https://doi.org/10.1007/s10664-020-09851-6>
- [18] IBM, “IBM Rational Change,” 2023, april 2023. [Online]. Available: <https://www.ibm.com/products/rational-change>
- [19] —, “IBM Rational Synergy,” 2023, april 2023. [Online]. Available: <https://www.ibm.com/products/rational-synergy>
- [20] V. Jacques, “PyGitHub,” 2007. [Online]. Available: <https://pygithub.readthedocs.io/en/latest/introduction.html>
- [21] M. Jaskolka and Gor-Marks, “McSCert/Model-Comparison-Utility: Model Comparison Utility (Version 1.4),” Apr. 2022. [Online]. Available: <https://doi.org/10.5281/zenodo.6410073>
- [22] M. Jaskolka, V. Pantelic, A. Wassyng, M. Lawford, and R. Paige, “Repository mining for changes in Simulink models,” in *Proc. 24th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MODELS)*. IEEE, 2021, pp. 46–57.
- [23] M. Jaskolka, V. Pantelic, A. Wassyng, R. Paige, and M. Lawford, “Repository mining for changes in Simulink and Stateflow models,” *Software and Systems Modeling*, Jun. 2023.
- [24] R. Jongeling, A. Cicchetti, F. Ciccozzi, and J. Carlson, “Co-evolution of Simulink models in a model-based product line,” in *Proc. ACM/IEEE 23rd International Conference on Model Driven Engineering Languages and Systems (MoDELS)*, E. Syriani, H. A. Sahraoui, J. de Lara, and S. Abrahão, Eds. ACM, Oct. 2020, pp. 263–273. [Online]. Available: <https://doi.org/10.1145/3365438.3410989>
- [25] H. H. Kagdi, M. L. Collard, and J. I. Maletic, “A survey and taxonomy of approaches for mining software repositories in the context of software evolution,” *J. Softw. Maintenance Res. Pract.*, vol. 19, no. 2, pp. 77–131, 2007. [Online]. Available: <https://doi.org/10.1002/smr.344>
- [26] M. Kajko-Mattsson, “A survey of documentation practice within corrective maintenance,” *Empir. Softw. Eng.*, vol. 10, no. 1, pp. 31–55, 2005. [Online]. Available: <https://doi.org/10.1023/B:LIDA.0000048322.42751.ca>
- [27] 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.
- [28] J. A. H. López and J. S. Cuadrado, “MAR: a structure-based search engine for models,” in *Proc. ACM/IEEE 23rd International Conference on Model Driven Engineering Languages and Systems (MoDELS)*, E. Syriani, H. A. Sahraoui, J. de Lara, and S. Abrahão, Eds. ACM, Oct. 2020, pp. 57–67. [Online]. Available: <https://doi.org/10.1145/3365438.3410947>
- [29] —, “An efficient and scalable search engine for models,” *Softw. Syst. Model.*, vol. 21, no. 5, pp. 1715–1737, 2022. [Online]. Available: <https://doi.org/10.1007/s10270-021-00960-4>
- [30] MathWorks Inc, “MATLAB & Simulink,” 2021. [Online]. Available: <https://www.mathworks.com/products/simulink.html/>
- [31] —, “Create block masks,” 2022, accessed Nov 2022. [Online]. Available: <https://www.mathworks.com/help/simulink/block-masks.html>
- [32] —, “Set model configuration parameters for a model,” 2022, accessed Nov 2022. [Online]. Available: <https://www.mathworks.com/help/simulink/ug/configuration-parameters-dialog-box-overview.html>
- [33] —, “Compare and merge Simulink models containing Stateflow,” 2023, accessed Mar 2022. [Online]. Available: <https://www.mathworks.com/help/simulink/slref/compare-and-merge-simulink-models-containing-stateflow.html>
- [34] —, “DocBlock,” 2023, april 2023. [Online]. Available: <https://www.mathworks.com/help/simulink/slref/docblock.html>
- [35] —, “Model Info,” 2023, april 2023. [Online]. Available: <https://www.mathworks.com/help/simulink/slref/modelinfo.html>
- [36] —, “Model comparison,” Accessed December 2022, 2022. [Online]. Available: <https://www.mathworks.com/help/simulink/model-comparison.html>
- [37] natedana, “How to copy a github repo without forking,” 2023, march 2023. [Online]. Available: <https://gist.github.com/natedana/cc71d496b611e70673cab5e8f5a78485>
- [38] V. Pantelic, A. Schaap, A. Wassyng, V. Bandur, and M. Lawford, “Something is rotten in the state of documenting Simulink models,” in *Proc. 7th International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*. SciTePress, Feb. 2019, pp. 503–510.
- [39] E. J. Rapos and J. R. Cordy, “Examining the co-evolution relationship between simulink models and their test cases,” in *Proc. 8th International Workshop on Modeling in Software Engineering (MiSE)*. ACM, May 2016, pp. 34–40. [Online]. Available: <https://doi.org/10.1145/2896982.2896983>
- [40] S. Romano, M. Caulo, M. Buompastore, L. Guerra, A. Mounsif, M. Telesca, M. T. Baldassarre, and G. Scanniello, “G-Repo: A tool to support MSR studies on GitHub,” in *Proc. 28th IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, Mar. 2021, pp. 551–555.

- [41] 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 2019*. IEEE, 2019, pp. 205–215.
- [42] 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 *Proc. 21st International Systems and Software Product Line Conference (SPLC)*, 2017, pp. 215–224.
- [43] S. L. Shrestha, "EvoSL: A Large Open-Source Corpus of Changes in Simulink Models & Projects," Apr. 2023. [Online]. Available: <https://doi.org/10.5281/zenodo.7806456>
- [44] S. L. Shrestha, A. Boll, S. A. Chowdhury, T. Kehrner, and C. Csallner, "50417/EvoSL-Tool: EvoSL: A Large Open-Source Corpus of Changes in Simulink Models & Projects," Jul. 2023. [Online]. Available: <https://doi.org/10.5281/zenodo.8111019>
- [45] S. L. Shrestha, A. Boll, T. K. Shafiu Azam Chowdhury, and C. Csallner, "EvoSL: A large open-source corpus of changes in Simulink models & projects (analysis data)," Jul 2023. [Online]. Available: [https://figshare.com/articles/dataset/EvoSL\\_A\\_Large\\_Open-Source\\_Corpus\\_of\\_Changes\\_in\\_Simulink\\_Models\\_Projects\\_Analysis\\_Data\\_/22298812](https://figshare.com/articles/dataset/EvoSL_A_Large_Open-Source_Corpus_of_Changes_in_Simulink_Models_Projects_Analysis_Data_/22298812)
- [46] S. L. Shrestha, S. A. Chowdhury, and C. Csallner, "DeepFuzzSL: Generating models with deep learning to find bugs in the Simulink toolchain," in *Proc. 2nd Workshop on Testing for Deep Learning and Deep Learning for Testing (DeepTest)*. ACM, May 2020, paper <http://ranger.uta.edu/csallner/papers/Shrestha20DeepFuzzSL.pdf>.
- [47] —, "SLNET: A redistributable corpus of 3rd-party Simulink models," in *Proc. 19th IEEE/ACM International Conference on Mining Software Repositories (MSR)*. IEEE, May 2022, pp. 237–241.
- [48] S. L. Shrestha and C. Csallner, "SLGPT: Using transfer learning to directly generate Simulink model files and find bugs in the Simulink toolchain," in *Proc. 25th International Conference on Evaluation and Assessment in Software Engineering (EASE), Vision and Emerging Results Track*. ACM, 2021, pp. 260–265. [Online]. Available: <https://doi.org/10.1145/3463274.3463806>
- [49] Software Heritage, "Faq—software heritage," 2023, accessed April 2023. [Online]. Available: [https://www.softwareheritage.org/faq/#42\\_Can\\_I\\_clone\\_a\\_repository\\_using\\_SWH](https://www.softwareheritage.org/faq/#42_Can_I_clone_a_repository_using_SWH)
- [50] D. Spadini, M. F. Aniche, and A. Bacchelli, "Pydriller: Python framework for mining software repositories," in *Proc. ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE*. ACM, Nov. 2018, pp. 908–911. [Online]. Available: <https://doi.org/10.1145/3236024.3264598>
- [51] M. Stephan, M. H. Alalfi, and J. R. Cordy, "Towards a taxonomy for Simulink model mutations," in *Proc. 7th IEEE International Conference on Software Testing, Verification and Validation (ICST) Workshops*. IEEE, Mar. 2014, pp. 206–215.
- [52] M. Stephan, M. H. Alalfi, J. R. Cordy, and A. Stevenson, "Evolution of model clones in Simulink," in *Proc. Workshop on Models and Evolution co-located with ACM/IEEE 16th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2013)*, 2013, pp. 40–49. [Online]. Available: <http://ceur-ws.org/Vol-1090/5.pdf>
- [53] E. D. Tempero, C. Anslow, J. Dietrich, T. Han, J. Li, M. Lumpe, H. Melton, and J. Noble, "The Qualitas corpus: A curated collection of Java code for empirical studies," in *Proc. 17th Asia Pacific Software Engineering Conference (APSEC)*, Nov. 2010, pp. 336–345.
- [54] M. Z. Trujillo, L. Hébert-Dufresne, and J. P. Bagrow, "The penumbra of open source: Projects outside of centralized platforms are longer maintained, more academic and more collaborative," *EPJ Data Science*, vol. 11, no. 1, pp. 1–19, 2022.