Software and Systems Modeling manuscript No.
(will be inserted by the editor)

SimIMA: A Virtual Simulink Intelligent Modeling Assistant

Simulink Intelligent Modeling Assistance through Machine Learning and Model Clones

Bhisma Adhikari · Eric J. Rapos · Matthew Stephan

Received: date / Accepted: date

Abstract Intelligent virtual model assistance is a key challenge in cultivating model-driven engineering proliferation and growth. Such assistance will help improve the quality of software models, support education for students learning modeling, and lower the entry barriers to new modelers. We present SimIMA, an intelligent modeling assistant for Simulink, which is an extremely popular modeling language in both industry and academia. SimIMA provides modelers with two different forms of data-driven guidance using a knowledge base of configurable repositories and sources. The first form of guidance, SimGESTION, suggests to modelers single-step operations they can perform on their models as they edit them in their modeling environment. These suggestions are based on the machine learning technique of ensemble learning through association rule mining and frequency classification. The second form of guidance, SimXAMPLE, presents modelers with similar/related Simulink systems for modelers to either insert directly into their environments or to view for inspiration. SimXAMPLE accomplishes this through model clone detection. To validate SimIMA, we conduct experiments using an established, open, and curated large set of Simulink models coming from a variety of application domains. Our results show that both of SimIMA's forms of guidance are inferring the appropriate model and element suggestions given SimIMA's knowledge base and that SimIMA is both scalable and efficient. Through our evaluation, SimIMA demonstrates a prediction accuracy of 78.86% for block-level suggestions and 82.04% for full system suggestions.

This material is based upon work supported by the National Science Foundation under Grant No. 1849632.

Bhisma Adhikari · **Eric J. Rapos*** · Matthew Stephan
Miami University
Department of Computer Science and Software Engineering
Oxford, Ohio, USA
E-mail: rapose@miamioh.edu

1 Introduction

Model-driven engineering (MDE) continues to see notable adoption in industry [37], educational contexts [17], and research [19]. As with any evolving and maturing field, MDE has certain challenges that it must overcome and milestones it must achieve to flourish and realize its full potential. One such example, identified at the "2017 Grand Challenges in Modelling workshop" and its subsequent publication [19], that MDE must address is the need for cognizance-based data-driven approaches to assist engineers while they are developing their modeling artifacts [21]. Such approaches have seen much success in traditional, source code, software engineering contexts, for example, those that provide coding assistance [11,51,52]. The same does not hold nearly as true for MDE. One promising approach to address this open MDE challenge is an intelligent virtual modeling assistant capable of providing data-based examples and/or guidance to engineers as they create their software models [21]. This will bring tangible and measurable benefits to MDE approaches. Mussbacher et al. [46] further discuss the need and landscape for intelligent modeling assistance, while also providing a reference framework, or guidelines, to those building intelligent modeling assistants (IMA).

In a New Ideas and Emerging Results paper, we presented our general high-level ideas for realizing an IMA that employs model clone detection/analysis, inference, and machine learning [67]. Our ideas included two forms of cognification and software modeling assistance.

The first was suggesting and presenting entire models to engineers that are similar to incomplete models they are currently developing for guidance or direct insertion. These suggested models come from analysis on a knowledge base consisting of software models from configurable sources and visualized to engineers in their native interface. The second was step-wise suggestions for engineers to consider during development, allowing them to visualize, analyze, and apply data-driven suggestions.

In this article, we describe our efforts in researching and developing those ideas. Specifically, we answer the following research questions by investigating and developing a research proof of concept in the form of a virtual Simulink intelligent modeling assistant (SimIMA),

RQ1 - To what extent can, and how accurate is, SimIMA in providing step-wise model element suggestions on configurable model sets?

RQ2 - Does SimIMA's use of model clone detection to discover similar complete subsystems for insertion/inspiration produce accurate/correct model suggestions?

While we aim to define our proof of concept in a language-agnostic generalizable fashion, we chose Simulink as our initial target language due to its popularity especially in the emerging areas of cyber-physical, embedded [53], and most recently, machine learning[1], systems. Additionally, there are many open and growing repositories with Simulink models including Matlab Central[2], SourceForge, and other corpuses [22]. We also consider and reference IMA guidelines, RF-IMA, proposed by Mussbacher et al. [46] where appropriate.

SimIMA is composed of two forms of modeling assistance, corresponding to our research questions, and enables engineers to request assistance in their native environments while developing their models. The first form of modeling assistance, corresponding to our first research question and termed SimGESTION, provides engineers with multiple suggestions for which Simulink block they should add next to their model based on a combination of techniques from machine learning, including classification based on association rule mining (ARM) and frequency matching. We evaluate SimIMA through empirical experiments using a curated [22] and independently validated [16] data set. The second form of modeling assistance, corresponding to our second research question and termed SimXAMPLE, allows engineers to 1) visualize similar models based on clone analysis and data inference with engineer-customizable repositories and examples, and 2) either select one of these similar model examples for direct insertion or display it adjacently for inspiration and guidance as they create and edit their models. We explicate our contributions as follows,

1. A model assistance approach that employs machine learning techniques on configurable model sets to give engineers step-wise guidance in the development of their systems.
2. A model assistance approach that uses model clone detection on configurable model sets to provide suggestions to engineers for inspiration and application.
3. A realization and demonstration of both of our approaches in a complete Simulink Intelligent Modeling Assistant, SimIMA, solution available to engineers for use in their native Simulink environments.
4. An evaluation of our approaches using a curated and established corpus of Simulink models that has been independently validated.
5. All of our data and artifacts posted on a public and persistent repository, allowing for reproduction and replication.

We begin with background information in Section 2. Following that, we provide an overview of SimIMA in Section 3 including a running example to help illustrate our work. Sections 4 and 5 present our approaches for SimGESTION and SimXAMPLE, respectively, including our design decisions, research hurdles, and user interface decisions for each. Section 6 presents our evaluation of each aspect of SimIMA. We follow this up with related work in Section 7, and provide our conclusions, threats to validity, and future work in Section 8

2 Background

Given the context of this article, we presume the reader has a foundational understanding of MDE. We begin with an introduction to recommender systems and their use in software engineering. We provide a brief background on Simulink, as it is the target language for our research proof-of-concept. We also outline model clone detection, as that is what SimXAMPLE employs to provide assistance. For machine learning background, we discuss the main concepts relevant to our SimGESTION assistance approach: classification, association rule mining, frequency matching, and ensemble approaches.

[1] `https://www.mathworks.com/help/stats/machine-learning-in-matlab.html`

[2] `https://www.mathworks.com/matlabcentral/`

2.1 Recommendation Systems

In this section, we discuss software engineering recommendation systems in general. In our related work section, we address related model assistance and completion work, specifically.

Software suggestion and completion can be considered a form of software engineering recommendation systems [55]. There are many successful examples of source code suggestion and completion systems including those that use matching algorithms [51], language-based techniques [52], and context of methods and call sites [11]. The Eclipse source code recommender facility is a prevalent code completion example that provides suggestions by analyzing API usage statistics [31]. While a modeling assistant is intrinsically different than all these approaches due to the graphical nature and context of software modeling, this and related research contain ideas and concepts we consider since our research accomplishes an analogous goal. Even though models typically can be serialized using a textual representation, editing and completion at the textual level would not be practical nor useful to a modeler developing their system. As such, a new paradigm is needed for intelligent modeling assistance that incorporates a graphical interface aspect into the suggestion mechanism.

Our research is based on using previous software examples. Bruch et al. identify three categories of data / information based on their work with source code [18]:

Frequency information is used to provide recommendations based simply on the number of occurrences of its use in example codebases.

Association Rules require finding all rules such that X implies Y, and then recommending Y whenever future instances of X are identified.

Matching Neighbours involves determining the context of a variable, searching for its use from example codebases, and synthesizing recommendations.

We employ aspects of all three categories in devising our approaches and SimIMA. In Bruch's case, they use a feature vector to form a structure. In our case, the structure is inherent in the model itself and forms the basis for comparison and suggestion. We determine the frequency of past use given a set of structural neighbours and also employ association rules. Our research on SimXAMPLE aligns also with Proksch's work, except in our case our matching neighbor algorithm employs model clone detection [51]. Robbes and Lanza demonstrated how using repositories and examples can improve code completion [54], which we contend holds true for MDE analogously.

While the topic of recommender systems in model driven engineering is quite new, Almonte et al. present a survey of current approaches [7]. In the survey they present an analysis of 66 papers that discuss the issues of model recommender systems in some form, including a classification of seven different types of recommender systems used in MDE applications, as well as various other classifications.

2.2 Simulink

We employ Simulink as the language for our research and approach/system development for multiple reasons. According to its website, Matlab Simulink is used at over 5000 academic institutions in a variety of STEM disciplines. Additionally, it has the most mature techniques for model clone detection [69], and is quite popular and growing in cyber-physical, embedded, and machine learning systems [53]. Simulink programs/models are created within the Matlab environment. Simulink models are data-flow models consisting of three levels of granularity: whole models, systems, and blocks. Models contain systems, and systems contain other (sub) systems, and blocks. Simulink blocks come from libraries and are connected by signal lines. An example Simulink model with contained subsystems, various blocks, and the connecting signal lines is in Fig. 1 with blocks and subsystems labeled by name. Simulink models have semantics and allow for parametrization and simulation. Many blocks have corresponding code that can be embedded on a variety of platforms. Modelers edit Simulink models through the Matlab environment by exploring systems and adding, modifying, and deleting blocks and lines. Developers can write their own Simulink applications, which include GUIs, task automation, and various other user-defined functions. Simulink blocks form the basis of the operations performed on the data being processed by the models/subsystems, and usually represent single operations such as mathematical operations, data processing, and logical comparisons. For the purposes of our research, we omit any analysis of custom block types and focus only on the standard Simulink block types.

2.2.1 Simulink Model Formats

Recent versions of Simulink produce models in an SLX format model, however within standard model repositories, significant numbers of legacy models exist in the older MDL format. Further, many model analysis tools, including leading clone detection tools, heavily rely on models in their MDL format, even necessitating a tool capable of automatically converting from SLX to

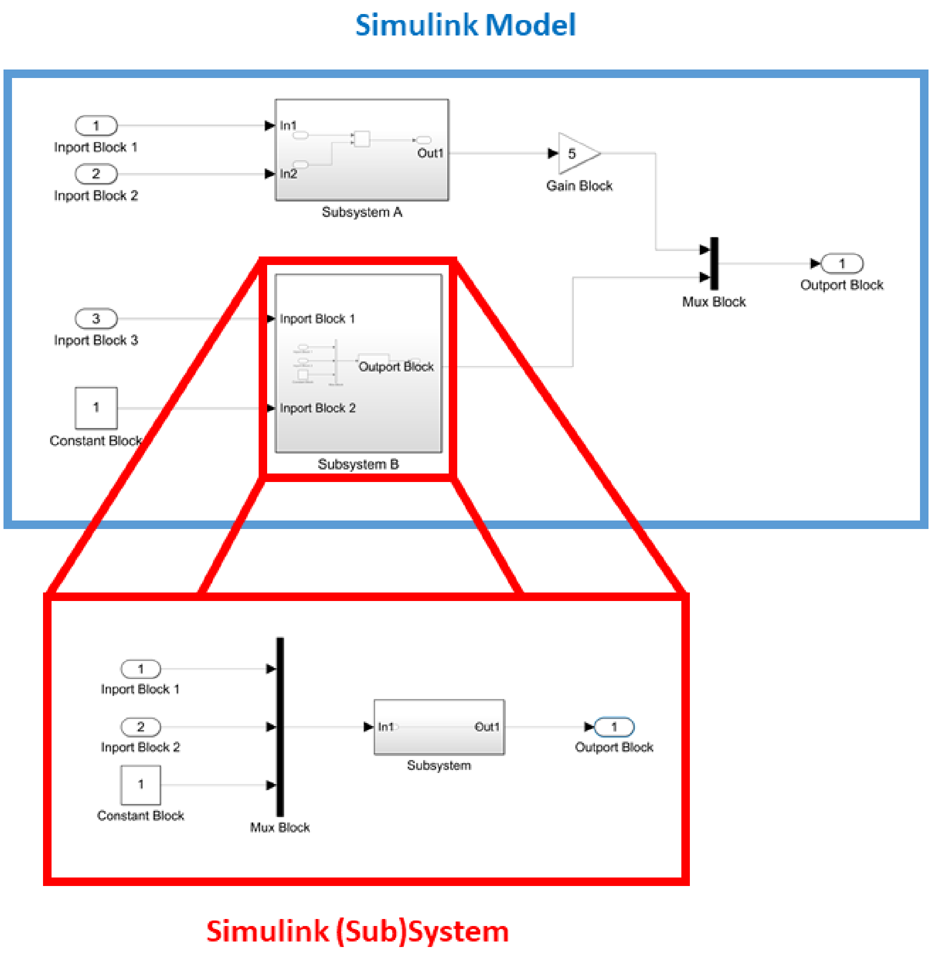


Fig. 1: An Example Simulink Model showing (sub)systems, blocks, and connecting signal lines

MDL formats [3]. In this section we briefly describe the differences between these formats.

The original, and still widely used, MDL format is a plain text representation of the model in a single flat file, where model properties are stored in a key-value pair format. The SLX format, which was introduced in the R2012b version of Matlab and Simulink, is a Mathworks proprietary format that stores model contents in an archive file containing mostly XML-based files stored in a hierarchical structure [3].

2.2.2 Matlab App Designer

Matlab App Designer[3] is the recommended way for building Matlab applications. Using App Designer is important to us because we want to provide engineers modeling assistance in their native environments to reduce disruption and encourage adoption. It provides a visual-driven approach for creating applications in which developers can layout components and customize behavior. It allows us to directly interact with the Simulink models and their underlying representations/data, while also facilitating external program calls.

[3] `https://www.mathworks.com/products/matlab/app-designer.html`

2.3 Model Clone Detection

Model clone detection performs model comparison analysis [69] to discover similar models according to some similarity definition [25]. Due to the graphical nature of models, techniques intended for traditional code clones, such as lexical and syntactic code clone detectors, are not suited to model clones and require model-specific techniques [6, 25, 71]. Model clone pairs can be clustered and classified into groupings known as model clone classes, characterizing a repeated modeling pattern [23].

There are different model clone detection techniques that work with a variety of model types. The most mature type of model clone detection is focused on Simulink models [25, 6, 50, 49], but emerging techniques for other model languages exist, including UML models [71, 10], and Stateflow [24, 40]. There are four different types of model clones. For our research, we are interested solely in near-miss (Type 3) clones as they include any exact (Type 1) or renamed clones (Type 2) and best align with our goal of providing suggestions of a similar, but not necessarily exact nature, while still providing multiple plausible and useful suggestions:

Type 3 - Near-Miss Clones are sets of model elements that are structurally different up to a specific threshold, ignoring visual differences, and names. Structural changes include different element orderings, the addition or deletion of elements, and more [6].

Type 4 clones, also known as Semantic Clones, are different source code fragments that perform the same function but are implemented using different syntactical representations; Type 4 clones are significantly more difficult to detect. [56]. The same is true for Type 4 model clones, which represent groups of model elements with the same semantic purpose, usually consisting of different blocks and representations; they are equally difficult to detect [66]. Due to the inability to incorporate a reliable Type 4 clone detector into our framework, we opted to focus on Type 3 clones.

For our research and application, we will be using the Simone Simulink model clone detector [6]. Simone detects type 1, 2, and 3 model clones by analyzing Simulink models' underlying textual representations. Simone filters, normalizes, and sorts these textual representations before performing text-based model clone detection, which has advantages over graph-based detection techniques [6]. It additionally clusters model clones into model clone classes. According to evaluations [70], Simone is the most adept Simulink model clone detector for detecting Type 3 clones. That, and Simulink's prevalence in emerging areas of modeling, supports our decision to use Simone for our initial realization and demonstration of our research.

2.4 Machine Learning

Machine learning is a diverse and evolving area. We focus specifically on the aspects we employ in our approach to step-wise suggestion, SimGESTION.

2.4.1 Classification

Classification is a form of supervised learning whereby new items (those being classified) are assigned a class label [39]. This assignment is based on a classifier that generalizes new instances based on past data. In the case of SimGESTION, we are considering a Simulink block, more specifically its block type, to be a new instance. The different Simulink block types are the potential classes of that new instance. Our classifier is based on a combination of factors we describe herein.

2.4.2 Association Rule Mining

Using association rule mining (ARM) is an established way of building a classifier [43]. Association rule mining involves devising rules of association/correspondence that satisfy some minimum confidence [5]. Simply put, ARM involves observing that when some set of elements, for example antecedents X and Y, are present, some other element, for example consequent Z, is also often present. The confidence is a number between 0 and 1 that represents the percentage of time that rule holds true in the rule derivation process. For our purposes, we are using association rule mining to determine the association of Simulink block types. For example, when blocks of type X and Y are present, how often block Z is present. These rules factor into our classifier when making our suggestions.

2.4.3 Frequency-and-Context-Based Learning

Another source of data for our classifier is using frequency and context information (FREQ). This is inspired by Baruch's research devising a code completion approach that considers both frequency of method calls and context [18]. Specifically, they count the total number of methods in a program and also consider context in the form of neighbours. While this is also a form of association, it differs from ARM in that 1) the new instance is allowed to be within the set of antecedents and 2) context (neighbors in this case) is considered in this form of learning rather than just mere presence.

2.4.4 Ensemble Learning

Ensemble methods in machine learning are those that construct multiple classifiers and combine them to perform classification on new instances [28, 74]. This is often done by weighting of the individual classifiers and by testing the ensemble classifier. There are many examples of classifiers being employed by analysts [59]. They often improve prediction and performance by reducing the risk of obtaining a local minimum, avoiding overfitting, and by having a better hypothesis discovered by the combination of two classifiers not originally possible with a single classifier. We employ an ensemble approach in SimIMA.

3 SimIMA Overview

In this section, we provide an overview of the architecture and main components of SimIMA. At the highest level, SimIMA is composed of two main modules: SimGESTION and SimXAMPLE. SimGESTION provides block-level suggestions for completing models, addressing our first research question. SimXAMPLE provides whole system examples to the user for inspiration and/or insertion. It addresses our second research question.

3.1 SimIMA Components

We define the four components of SimIMA consistent with the RF-IMA framework [46]: the assistant, the data acquisition/production layer, the context shadow, and the optional adaption.

1. The *assistant* for SimIMA is realized as a Simulink application that provides graphical recommendations to engineers directly into their Simulink development environment and also contextualizes the information showing where potential suggestions can be applied by SimIMA.
2. In SimIMA, our *data acquisition/production layer*, which provides access to and allows configuration of different sources, is facilitated through our repository configuration within our different Simulink menus that we customized using App Designer. These menus connect to our external applications that perform the data acquisition and suggestion generation.
3. SimIMA's *context shadow*, which captures the context and current activity of the engineer, is the Simulink environment itself whereby we capture context information. This includes the engineer's current system under development (SUD), the most recently clicked block (MRCB) by the engineer, information about the MRCB, and more.
4. While optional, we also have an *adaption* component, which allows engineer feedback to change both data and context. SimXAMPLE allows the engineer to regenerate their data after seeing suggested, whole, subsystems by fine tuning both the parameters of the search and the data. While SimGESTION does not incorporate feedback at this point, it is something we consider future work.

The creation of the RF-IMA framework, along with its evaluation methods [45,46] came from a need for a standard which did not yet exist. While it may be too soon to determine whether it will become the de facto standard for defining and evaluating intelligent modeling assistants, it currently serves this purpose. Further, the authors responsible for its creation include leaders in the fields of software modeling and modeling assistance, and this added confidence in our usage of the RF-IMA framework in the creation and evaluation of SimIMA. While there may later be other frameworks to reference, at the time of this research, it is the best benchmark we can measure against.

We present SimIMA's architecture in Fig. 2, which involves a combination of components internal and external to Matlab. For SimXAMPLE, engineers can request model design assistance through our Simulink custom menus. We use App Designer to present suggestions to the engineers and Matlab script files to connect, configure, and execute Simone, which is external to Matlab. For SimGESTION, we suggest blocks within the actual model development interfaces. Our machine learning algorithms are encoded within Matlab scripts. In both cases, it begins with a work-in-progress model under development (MUD) by an engineer who requests assistance. In the case of Simulink, where models contain one or more (sub)systems, the MUD contains a (sub)system Under Development (SUD), potentially along with other (sub)systems. We discuss our research and approach using some Simulink terms and concepts, however, our approaches and ideas are intended to be transferable to other modeling languages unless we specify otherwise.

SimIMA comes with 2 standard repositories that can be selected/deselected. Engineers are able to add custom model repositories through our interface, which corresponds to the recommendation from Dyck et al. to allow for model recommenders to query multiple sources [29]. Currently, for demonstration purposes, the included "standard" repositories are composed of a variety of Simulink example models from the "Other (Unidentified)" category from the Chowdhury et al. [22] corpus, which we discuss in more detail later. However, engineers can link to their own in-house model projects, domain exemplars, and other repositories as they desire. We plan to extend the standard/default repositories in the future to link SimIMA to online model repositories, such as Matlab Central, the Model Clone Portal (MoCoP) [12], and others including those identified by Chowdhury et al [22]. We opted for the "Other (Unidentified)" category for illustrative purposes solely because we were not building models in any specific domain and felt that a broader selection of

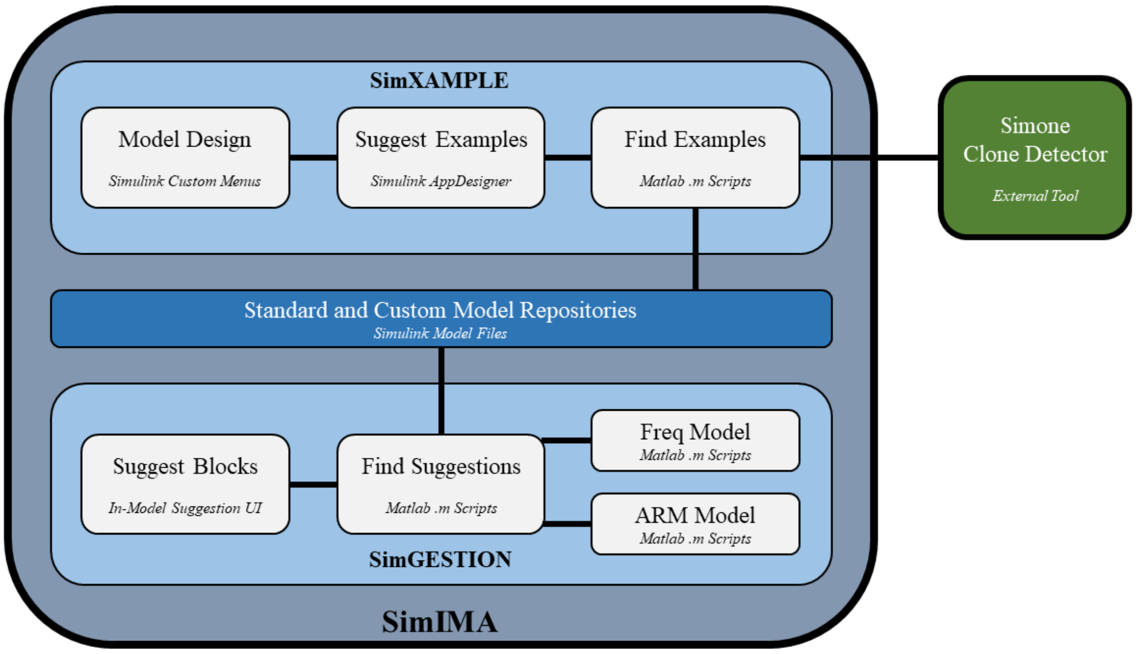


Fig. 2: SimIMA Architecture Overview

uncategorized models would provide the most generalizable suggestions.

The latest version of SimIMA's source is available on our public and persistent repository [2].

3.2 Running Example - ExampleSUD

In order to best illustrate the application of SimIMA to systems under development in the remainder of the paper, we use a running example throughout. Our running example, ExampleSUD, begins as a relatively new system undergoing development. We develop ExampleSUD first using SimGESTION and then SimXAMPLE once it is more advanced. This is not meant necessarily to showcase a standard series of events, but rather to illustrate the two assistance approaches in one running example to assist the reader's understanding. Our example employs the standard repositories that come standard in SimIMA to facilitate reproduction and replicability. The initial composition of ExampleSUD, as seen in Fig. 3, contains a single inport block, which is a typical starting point for most systems. The reason this relatively simple example was chosen is to show that even a single block is sufficient to begin receiving suggestions from SimGESTION. Additionally, a more complex example would require an advanced understanding of both Simulink and the target domain,

Fig. 3: Initial Composition of ExampleSUD

however, beginning with a simple import block allows the model to be created entirely by suggestion.

4 SimGESTION

This section describes our research in developing an approach for step-wise suggestions in Simulink, SimGESTION. We begin with a presentation of our approach, including a discussion on our design considerations. We follow this with our resulting Simulink realization and user interface, and illustrate it through our running example.

4.1 Approach

SimGESTION employs an ensemble method approach for classification due to the benefits afforded by combining multiple classifiers. Based on a review of existing machine learning methods and related work, along with an analysis of the type of data available and intended suggestions, we determined that ARM and FREQ

Table 1: Example ARM Matrix

	Gain	**Inport**	**Outport**	**Scope**	**Sine**	**Sum**
S1	1	0	1	0	0	1
S2	0	1	0	1	0	1
S3	1	0	1	0	1	0
S4	1	0	1	1	0	1

classification techniques were both appropriate and logical in our context. While other techniques may also be relevant, these were chosen for their applicability. Ultimately, a comparison with other similar techniques would not likely have yielded significantly different results, and falls outside of the scope of this research. As per ensemble learning, SimIMA derives classifiers using ARM and FREQ separately, and weights and combines those classifiers. This happens each time the engineer customizes their repositories (data). We now present our ARM and FREQ algorithms, followed by the ensemble application process. While we present our empirically derived parameter/tuning values in this section, we provide the details of that derivation in our evaluation section, Section 6. We make all of our algorithms, scripts, experiments, and data available online for reproduction and replication [2,1].

4.1.1 ARM Classifier and Suggestions

SimIMA's first step in ARM is to create a two-dimensional matrix where columns represent Simulink block types and rows represent (sub) systems. Within the matrix, each system row entry will indicate whether that specific block type is present in that system. Consider the following illustrative example of 4 systems in the training data and corresponding matrix in Table 1. A '1' in a cell indicates the presence of a block of that type in that respective system. To gather this information, we consider each repository configured by the engineer. For each repository, we iterate through each model and its respective subsystems, noting which block types are present and which are not. When we encounter a block type not yet in the matrix, we simply add a new column to the matrix, and update the entries for that new column accordingly. This allows us to establish our ARM antecedents-and-consequent rules/relationships accordingly. The example matrix corresponds to the following systems containing the listed blocks,

S1 : ["Sum", "Gain", "Outport"]
S2 : ["Inport", "Sum", "Scope"]
S3 : ["Gain", "Outport", "Sine"]
S4 : ["Gain", "Outport", "Scope", "Sum"]

To produce suggestions, we first establish the context of the assistance request, specifically, the SUD. As part of the context, we consider all block types present in the SUD to be the antecedents, including the MRCB. During our empirical experiments optimizing our classifier using grid search with nested 10-fold cross validation, we found loosening the percentage of required antecedents in a row yielded better performance. We found that requiring 87 % or more of the antecedent blocks be present provided the best results. We call this fraction i.e. 0.87 the ANT_REDUCTION. We then traverse the ARM matrix and for any block type that is present as a consequent to our antecedents, we calculate the confidence. Our confidence is measured as a fraction of the number of rows that have our antecedents and consequent over those rows in the matrix that have just the antecedents. When that confidence is non-zero, we consider it as a potential suggestion. After we complete this, we sort the suggestions by their confidence for presentation purposes.

To illustrate this process with an example, let us assume that our ARM model is trained on the data presented in Table 1. Also, let us assume that the current SUD contains only "Gain" and "Outport" block types. To simplify the example, let us set ANT_REDUCTION to 1. The ARM model computes confidence values for all block-types present in the ARM matrix that are missing in the current SUD as follows, where the numerators and denominators represent the number of rows in the table containing at least those blocks:

$$C_{Inport} = \frac{Gain,\ Outport,\ Inport}{Gain,\ Outport} = \frac{0}{3} = 0$$
$$C_{Scope} = \frac{Gain,\ Outport,\ Scope}{Gain,\ Outport} = \frac{1}{3} = 0.33$$
$$C_{Sine} = \frac{Gain,\ Outport,\ Sine}{Gain,\ Outport} = \frac{1}{3} = 0.33$$
$$C_{Sum} = \frac{Gain,\ Outport,\ Sum}{Gain,\ Outport} = \frac{2}{3} = 0.66$$

The ARM Model filters out any suggestions with a confidence equal to zero and then sorts the suggestions based on their confidence values. Ties, if any, are broken alphabetically by suggestion's block-type. For our example, the model then returns the final sorted list of suggestions as [Sum (0.66), Scope (0.33), Sine (0.33)].

4.1.2 FREQ Classifier and Suggestions

In employing frequency and context data, we were inspired by related source code work that considers method frequency [18]. For this classifier, SimIMA creates a nested-map data structure that summarizes the training data. We present an illustration of a contrived instance of that data structure in Listing 1 consisting of only two block types. Looking first at the 'Sum' component on line 3, the 'count' represents the total number of Sum blocks in the training set. The 'src' is the total number of blocks connected to the src port of 'Sum' blocks in training set. This tally is further broken down into the 'details' component. The 'dst' component contains the analogous information for all the Sum blocks in the training set. The 'both' component contains the summation of both 'src' and 'dst'. This pattern continues on line 32, with the 'Gain' block representing all Gain blocks in the training set. There is one such entry for each type of block SimIMA discovers during its repository (data) analysis. This nested map is important in the frequency analysis as it provides a complete representation of all blocks present in the repository, forming the basis of the classifier. The nested map is encoded in a Matlab Structure Array[4], which appears similar to a JSON representation.

```
 1  {
 2      'Sum': {
 3          'count': 304,
 4          'src':{
 5              'count': 120,
 6              'details': {
 7                  'Sum': 55,
 8                  'Gain': 21,
 9                  'Inport': 44,
10              }
11          },
12          'dst':{
13              'count': 167,
14              'details': {
15                  'Sum': 35,
16                  'Gain': 13,
17                  'Inport': 32,
18                  'Outport': 87,
19              }
20          },
21          'both':{
22              'count': 287,
23              'details': {
24                  'Sum': 90,
25                  'Gain': 34,
26                  'Inport': 76,
27                  'Outport': 87,
28              }
29          },
30      },
31      'Gain': {
32          'count': 434,
33          'src':{
34              'count': 42,
35              'details': {
36                  'Sum': 15,
37                  'Gain': 23,
38                  'Inport': 4,
39              }
40          },
41          'dst':{
42              'count': 107,
43              'details': {
44                  'Sum': 35,
45                  'Gain': 13,
46                  'Inport': 32,
47                  'Outport': 27,
48              }
49          },
50          'both':{
51              'count': 149,
52              'details': {
53                  'Sum': 50,
54                  'Gain': 36,
55                  'Inport': 36,
56                  'Outport': 27,
57              }
58          },
59
60      },
61  }
```

Listing 1: Example of Frequency Data

When suggestions are requested by the engineer, SimIMA captures the MUD and the MRCB's block type, T. SimIMA then derives a classifier using two different calculations. Firstly, it finds all blocks that are connected to T-type blocks as destination blocks. It then calculates a confidence value for each potential suggestion based on how often that suggestion was found as a destination block. Secondly, it also considers context in the form of neighbours. That is, it considers all blocks that are neighbours (sources or destinations) of T-type blocks. SimIMA once again calculates the confidence for each suggestion by seeing the frequency of that suggestion in all neighbours of T-type blocks. In deriving a total classifier for FREQ, we empirically derived a weighting of the suggestions based on MRCB and its neighbours. Specifically, we found that weighting destination alone at 0.9 and neighbours at 0.1 yielded the best accuracy.

4.1.3 Ensemble Classifier and Suggestions

After deriving the separate classifiers from ARM and FREQ whenever there is a change in the repositories, we employ an ensemble learning methodology to create one single classifier. Our algorithm merges the suggestions together from ARM and FREQ using our empirically derived weights to calculate a "total" confidence consistent with other ensemble methods that combine classifiers by "taking a (weighted) vote of their predictions", which was informed by the model presented by Dietterich [28]. We illustrate this in pseudocode in Listing 2. The key aspects are our use of weights for calculating confidence on lines 9 and 16 for ARM and FREQ, respectively. Confidence for

[4] `https://www.mathworks.com/help/matlab/ref/struct.html`

a suggestion starts at zero and is grown only when suggested by ARM or FREQ and weighted according to their respective weights. Our empirical derivation, which we describe in Section 6, had us set these weights as 0.3 for ARM and 0.7 for FREQ.

```
FUNCTION mergeSuggs(suggsArm, suggsFreq,
    ↪ weightArm, weightFreq)
suggs = new collection
For each suggestion SA in suggsArm:
    If there does not exist a suggestion S in
        ↪ suggs such that S.blockType == SA.
        ↪ blockType:
        Create suggestion S
        S.blockType = SA.blockType
        S.confidence = 0
        Add S to suggs
    S.confidence += SA.confidence * weightArm
For each suggestion SF in suggsFreq:
    If there does not exist a suggestion S in
        ↪ suggs such that S.blockType == SF.
        ↪ blockType:
        Create suggestion S
        S.blockType = SF.blockType
        S.confidence = 0
        Add S to suggs
    S.confidence += SF.confidence * weightFreq
Sort suggs by confidence -- high to low
Return suggs
```

Listing 2: Psuedocode for Ensemble Classifier Creation

4.1.4 Performance Tuning

In general, training these prediction classifiers involves loading the Simulink models from the standard and custom repositories in Simulink and extracting the information needed by SimGESTION. In our experiments and experience, it takes roughly less than a second per Simulink model to train the two prediction classifiers. This can vary depending on the size of the models. To speed up the process, we employed a number of optimizations since this performance would not scale well for large model repositories.

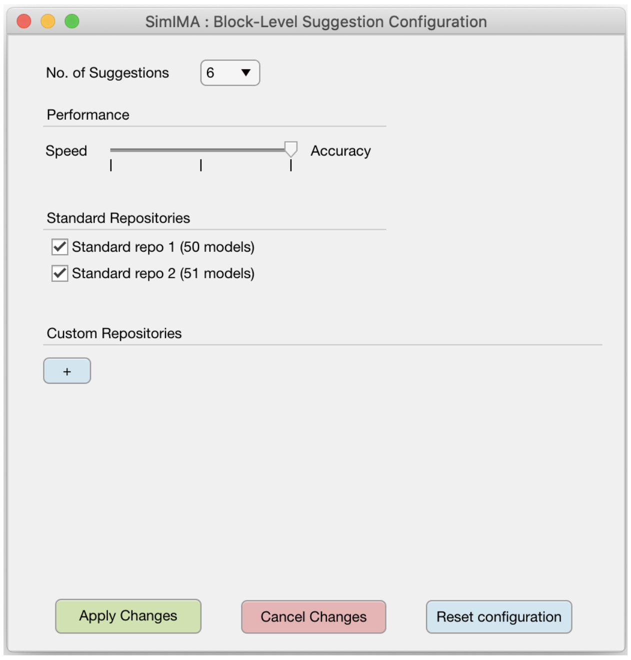
Firstly, we conduct "preliminary training" on the standard repositories. Specifically, we train both ARM and FREQ on those repositories and in all 3 permutations of both being selected, or just one of them being selected. Secondly, we employ a cache-like approach. For ARM and FREQ, we maintain a hash of the table/matrix and the frequency data, respectively. When using SimIMA to train ARM or FREQ, we first compute the hash value of the model file and see if that data is available in the corresponding cache. If so, we use that information to prevent having to retrain on data that already exists, thus enhancing performance when possible. Lastly, we perform classifier merging within ARM and within FREQ by considering the standard and custom repositories separately and merging after. This occurs each time the engineer makes a change in their selection of repositories. SimGESTION begins by loading in the appropriate preliminary trained classifiers of the standard repositories from our first optimization. It then compares the model files in the new custom repositories, and if it finds differences, then it updates the classifiers. It then performs classifier merging of the standard repositories classifier and the custom repositories classifier. For FREQ, this involves updating all the appropriate data. For ARM, we concatenate the two matrices into one. For context, details about the size and diversity of block types in the standard repositories are presented in Section 6.1. The full repositories are available within our evaluation data [1].

4.2 User Interface

To allow for engineer customization of data and learning parameters, we used Simulink App Designer to implement a SimIMA Block-Level Suggestion Configuration wizard. We illustrate this wizard in Fig. 4. We included the call to the wizard in the same model context menu as all other SimIMA commands. Within the wizard, engineers are able to adjust the number of suggestions shown by SimIMA, choose which of the standard repositories to include, add their own custom repositories, and adjust the performance of the recommendation. The performance slider allows the user to choose between speed and accuracy with 3 fixed positions. Prioritizing speed uses only the FREQ model because it produces suggestions through a lookup of its nested-map data. Thus, its suggestion-retrieval time is independent of the training data size, that is, O(1). This is in contrast to the retrieval time of ARM, which grows with the training data size. Prioritizing accuracy uses the ensemble model. The balanced option applies only the ARM model for a mix of speed and accuracy. While we have not completed a full analysis of speed vs. accuracy, these three settings allow for some customization by the user based on our initial comparisons. Any time an engineer changes the repository, it triggers a regeneration of the classifiers as appropriate/needed.

4.3 ExampleSUD

We start by querying SimGESTION on the initial ExampleSUD, containing the lone inport. This request is made by right clicking the block to reveal its context menu, to which we have added an option to have


SimIMA : Block-Level Suggestion Configuration

No. of Suggestions: 6

Performance

Speed — Accuracy

Standard Repositories

- ✓ Standard repo 1 (50 models)
- ✓ Standard repo 2 (51 models)

Custom Repositories

Fig. 4: SimIMA Block-Level Suggestion Configuration Wizard

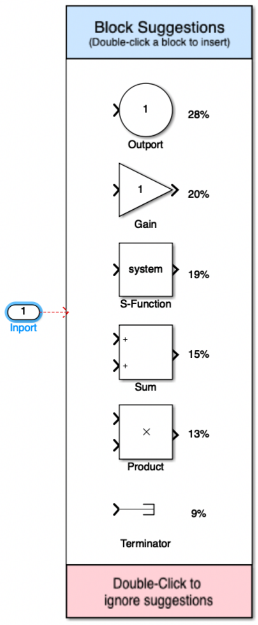


Fig. 5: Initial Suggestions for First Insertion into ExampleSUD

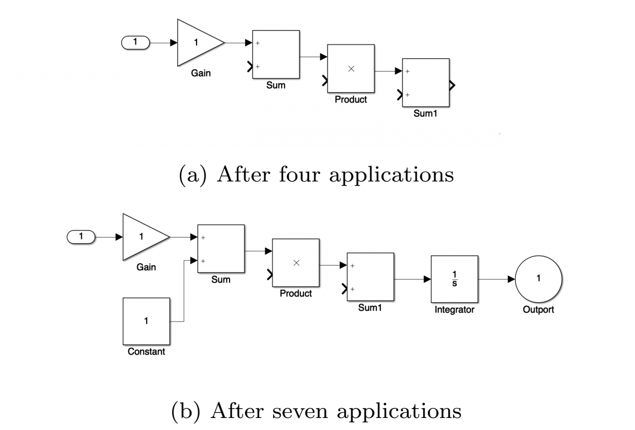


Fig. 6: Interim views of ExampleSUD after repeated applications of SimGESTION

SimGESTION produce suggestions. This provides the user with the top suggested, six (by default) blocks that we could connect. We illustrate the in-model suggestion view for this initial query in Fig. 5. Beside each block we see the confidence in that particular suggestion. For this running example, we opt to select the top most ranked non-terminal block to continue building the model in a step-wise fashion, with the most recently inserted block acting as the MRCB for the next suggestion. Following this process for four insertions, we produce the interim model in Fig. 6a. Continuing to apply a total of seven suggestions, we produce the interim model shown in Fig. 6b. SimGESTION can be applied continuously in this manner as long as the engineer wants to keep suggesting blocks to insert. Using a different MRCB will yield different suggestion results. Suggestions of where to connect the suggested blocks are not made. Instead, SimGESTION suggests a block that should connect to the MRCB, but the exact connectivity is left to the user to decide.

5 SimXAMPLE - Complete Model Examples

In this section, we present an overview of the SimXAMPLE process followed by a presentation of its phases, including our research hurdles and resulting process implementations. While our implementations are Simulink specific, our general process/concept is aimed to be applicable to any language that has type-3 model clone detection capabilities. The SimXAMPLE process, which we illustrate in Fig. 7, infers and visualizes model suggestions to engineers through a four-phase process: model clone detection, subsystem recommendation, candidate selection, and application to the engineer's model. At the highest level, it begins with a MUD. It ends with that model being updated/replaced, depending on the engineer's preference. More specifically, the input to this process is any MUD that the engineer is currently developing. When an engineer is unsure of how to complete an SUD within the MUD, they can use SimXAMPLE to find examples of other models that have similar initial structure. This allows them to comprehend how other modelers have completed similar systems, and optionally insert and/or merge them into their implementations. Additionally, secondary SimXAMPLE use cases include when the engineer has just completed creating a subsystem, or is evaluating an existing subsystem, and wants to compare visually their subsystem to other similar design alternatives, potentially for optimization or verification. The significant contributions of SimXAMPLE over the traditional application of model clone detection are in the configuration, presentation, and application of model clones, all of which are included in the default use of SimXAMPLE and described in this section. We now describe each phase, first conceptually, followed by how we realized it for SimXAMPLE specifically.

5.1 Phase 1: Clone Detection

The input to this first phase is the SUD. Its output is model clone detection data that are able to be interpreted by the modeling assistant in such a way that the assistant can provide recommendations to the engineer. The essence of this phase is the inclusion of model clone detection directly into the targeted environment. That is, a model clone detector capable of detecting Type 3 model clones must be "hooked" into the engineers' interface. This includes being able to receive sufficient SUD data such that the model clone detector can perform its analysis. Additionally, the engineer must be able to specify the knowledge base, in the form of repositories and sources, that the intelligent modeling assistant should have the model clone detector consider when inferring example model clones. Once the clone detector is configured visually in this manner by the engineer, the modeling assistant must be able to conduct the model clone analysis and inference without requiring the engineer to leave their interface as per established responsive requirements when implementing model recommenders [29].

5.1.1 Phase 1 Implementation

To achieve and realize this phase, we developed a user interface using AppDesigner. We have our custom application available via the menu of any Simulink model or subsystem. An engineer interested in a suggestion from SimXAMPLE can select the menu item from the menu of their SUD and be presented our phase 1 SimXAMPLE interface, or "landing page". We present an example of the SimXAMPLE landing page later during our running example in Fig. 11.

As we illustrate on the left of Fig. 11, the engineer is prompted to select the knowledge base repositories that SimXAMPLE will search and consider for clone analysis along with the SUD. Based on the repositories selected by the engineer, SimXAMPLE invokes Simone's cross-clone functionality. It also configures its clone similarity thresholds and other settings. Based on existing work evaluating Simone's ideal clone detection settings [70], we set Simone to use a 30% difference (70% similar) threshold by default, which finds model clones that are 30% different or less according to its algorithm. This difference threshold has been previously established and evaluated to find models that are related and useful clones [70]. Engineers can change this difference threshold according to their preference via the radio button we showcase in Fig. 11 by selecting Very ($> 90\%$), Somewhat ($> 80\%$), or Less similar ($> 70\%$), which we termed as such for user friendliness. This adheres to Dyck et al. recommendation pertaining to "allowing multiple recommender strategies" [29]. The cross-clone function performs model clone detection on the SUD against/across the complete set of models in all selected repositories. That is, it looks for clones of the SUD within the selected repositories, but will not find clones among the repositories themselves, focusing only on clone pairs containing the SUD.

The clone detection process is conducted in the background by SimXAMPLE. This use of clone detection on only the SUD rather than the complete model allows SimXAMPLE to focus on a local context rather than being obfuscated by model content not relevant to the current design context due to the nested clone problem [6]. This problem is an interesting research challenge we had to address. Specifically, Simone reports only the outermost model clones that meet its threshold and not any inner model clones. This is an issue because, under standard circumstances, Simone takes as input two Simulink models and detects model clones at the system level. If we were to pass in the MUD containing the SUD, Simone may identify system clones at a higher level/hierarchy than the SUD within the MUD, as seen in Fig. 8a. Thus, to avoid the nested clone problem, we do not treat the actual MUD as input. Instead, we create a temporary MUD that encapsulates and contains the SUD as the top-level system, while maintaining all the subsystems and elements contained within/below the SUD, as shown in Fig. 8b. We then run cross-clone detection with that temporary MUD. Once SimXAMPLE includes, configures, and executes the model clone detector, it must interpret and process the results to provide recommendations to the engineers, which is the focus of the next phase.

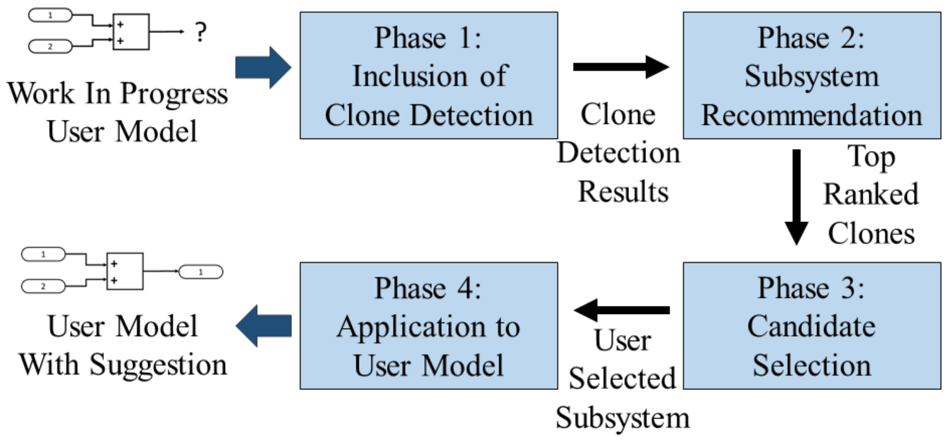
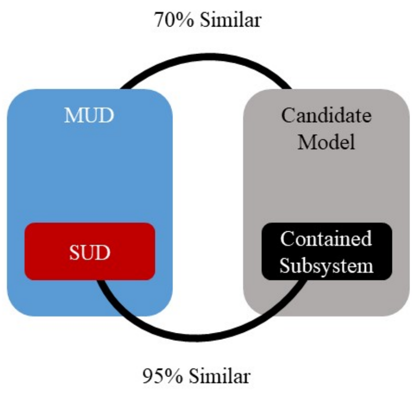
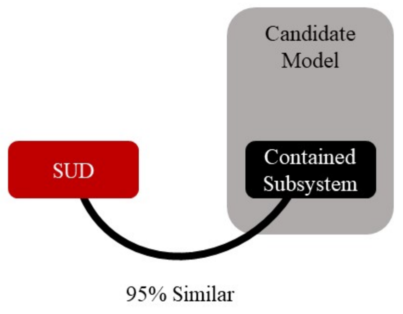

?
Work In Progress User Model
Phase 1: Inclusion of Clone Detection
Clone Detection Results
Phase 2: Subsystem Recommendation
Top Ranked Clones
Phase 3: Candidate Selection
User Selected Subsystem
Phase 4: Application to User Model
User Model With Suggestion

Fig. 7: SimXAMPLE Process Overview



(a) Example of Nested Clone Problem: Only the top-level (70%) match would be reported



(b) Solution to the Nested Clone Problem: By using the SUD as input, the best match is found

Fig. 8: The Nested Clone Problem

5.2 Phase 2: Subsystem Recommendation

This phase takes as input the tailored model clone detection results from the prior phase. This phase has the output of a list of specific (sub)system recommendations. The model clone detection results must be interpreted by a modeling assistant in such a way that only the most similar/related model clones are inferred and visualized to the engineer since that is what most likely mirrors their current intent. Depending on the nature of the model clone detection report and results, this involves various forms of interpretation and processing. After completing that interpretation and processing, the modeling assistant must present these results in a responsive non-blocking manner [30] that is natural and organic to the engineer's processes.

5.2.1 Phase 2 Implementation

Given the fairly extensive model clone detection report produced by Simone, SimXAMPLE must interpret and visualize only the most related and similar type 3 model clones to the engineer for consideration. SimIMA relies only on the serialized clone detection reports, such as the example below, for identifying the potential suggestions. However, it uses the internal Simulink model representations when displaying or applying the suggestions in later phases.

Listing 3: Sample Simone Report

```
<clones>
    <systeminfo processor="nicad3" system="mdl-
        ↪ file" granularity="systems-sort-blind
```

```
            ↪ " threshold="30%" minlines="10"
            ↪ maxlines="20000"/>
    <cloneinfo npcs="23" npairs="10"/>
    <runinfo ncompares="180" cputime="29160"/>
    <clone nlines="55" similarity="70">
        <source file="mdl-file/sud.mdl" startline
            ↪ ="2" endline="83" pcid="1"></
            ↪ source>
        <source file="mdl-files/repo1/
            ↪ adder_in_subsystem.mdl" startline=
            ↪ "1116" endline="1228" pcid="16"></
            ↪ source>
    </clone>
     ...
    <clone nlines="44" similarity="79">
        <source file="mdl-file/sud.mdl" startline
            ↪ ="2" endline="83" pcid="1"></
            ↪ source>
        <source file="mdl-files/repo1/
            ↪ adder_without_scope.mdl" startline
            ↪ ="1058" endline="1120" pcid="18"><
            ↪ /source>
    </clone>
</clones>
```

There are a number of considerations in deriving suggestions, including querying, ranking, and filtering [29]. Our initial implementation employs ranking suggestions based on their similarity to the SUD according to Simone's algorithm. To facilitate SimXAMPLE's suggestions of the most similar model systems, we implement filtering and sorting of the model clone detection results. The report from Simone's cross cloning feature is an XML report containing pairs of clones, with each pair containing the SUD as one of its elements, and supplemental data about the clone pairs. We provide an excerpt of such a report in Listing 3 to help us explain our interpretation and processing. The first step in calculating the suggestions is parsing this report to create an internal representation within Simulink for further processing.

In parsing the report, SimXAMPLE gains an internal representation that consists of a list of ***Clone*** objects, each with

- *nlines*: the number of source lines contained in the clone
- *similarity*: the percentage of similar lines between the clone pair
- *source1*: a ***Source*** object representing the SUD
- *source2*: a ***Source*** object representing the potential suggestion system

SimXAMPLE sorts and filters this clone list to provide optimal complete-system suggestions. First, SimXAMPLE sorts the list in decreasing order of similarity. It then filters that sorted list to provide the maximum number of suggestions, which is an engineer-configurable value, set by default to 10. Our filtering algorithm removes any clone pairs that have 100% similarity, as identical clones are not useful suggestions. It then removes the suggestions that exist after the maximum suggestion limit. The resulting list contains at most 10 clone pairs, which form the suggestions for completing the SUD.

For each suggestion, SimXAMPLE creates an internal model file and an image file for use in suggestion processing and visualization. It stores a complete ***Suggestion*** object, which has an internal representation consisting of

- *similarity*: the similarity to the SUD
- *source*: a ***Source*** object representing the potential suggestion system, corresponding to *source2* in a ***Clone*** object
- *mdlFile*: the file path to the MDL file containing the subsystem
- *imgFile*: the file path to the image file representing the system for displaying in the next phase
- *rank*: the numerical ranking of the suggestion

SimXAMPLE passes the list of ***Suggestion*** objects to the third phase for visualization to, and eventual selection by, the engineer.

5.3 Phase 3: Candidate Selection

This phase takes as input the inferred system recommendations that must be visualized to the engineer for them to browse. It ends with output representing an engineer choice/selection of which system they want to load for inspiration or direct application in to their model. This ability to directly insert and apply recommendations from a modeling assistant is consistent with the guidelines from Dyck et al [29]. We included the customization feature in SimXAMPLE to allow engineers to alter/update the suggestions through selecting alternative repositories, adding engineer-defined repositories, or adjusting the similarity threshold. They can then "refresh" the results through the corresponding button on the landing page. This causes the whole process to begin anew with new parameters. The SimXAMPLE interface is interactive and any changes to these parameters are reflected in the displayed results responsively and in real time.

5.3.1 Phase 3 Implementation

As we illustrate on the right of Fig. 11, SimXAMPLE presents the engineer with the two most similar and related suggestions at first. SimXAMPLE represents and visualizes the inferred systems to the engineer directly in the SimXAMPLE interface within Matlab by loading the model containing the systems and capturing an image representation at the appropriate hierarchy level of the respective inferred system suggestions. The interface visualizes each two options side-by-side with the similarity value of each candidate. The engineer has the ability to browse through pairs of suggestions, sorted by similarity ranking, using navigational buttons. SimXAMPLE displays the 10 most similar suggestions. An essential feature of our SimXAMPLE tool interface is the ability to select the desired suggested system directly in the interface by clicking on its representative image. This loads the model for further processing and prepares it for the next phase. This chosen (sub)system, which we refer to herein as the System From Suggestion (SFS), has already been loaded into Simulink's memory by SimXAMPLE during the previous phase, allowing SimXAMPLE to utilize it in the next phase, efficiently and responsively.

5.4 Phase 4: Application to User Model

The input to this final phase is the engineer-selected system they wish to load for inspiration and/or direct insertion into the modeling environment. There must be enough (meta)data about the system such that a modeling assistant can insert the system into the engineer's environment as correctly as possible, given the context. This includes any connections that are being replaced, or those that are implicit. The output for this phase and the process as a whole is an updated interface and model consistent with the engineer's expectations based on their selected suggestion.

5.4.1 Phase 4 Implementation

With the SFS selected by the engineer via the SimXAMPLE interface, the final step is the application of the SFS to the SUD. Rather than the simplified approach of merely replacing the entire SUD with the contents of the SFS as is, it is necessary for us to consider the impact of the replacement on the MUD as a whole. This introduces potential MUD variation that may occur due to potential mismatches in the block-port signatures of the SUD. Since we are using Type 3 model clones as our data, it is possible that the SFS has a different number of ports, such as inports, outports, or a combination of both, than the SUD. Thus, a direct replacement may cause inconsistencies. There are multiple possible approaches that we considered for SimXAMPLE to perform a replacement, which we categorize as follows,

1. direct copy of SFS into SUD
2. automatic merging of SFS into SUD
3. intelligent copy of SFS into SUD with engineer intervention

The first option can lead to unconnected ports, lost information, or model-build errors if done naively by a model assistant. For these reasons, we determined it not to be a viable solution on its own. The second option involves merging the two systems automatically to maintain the signature of the SUD while applying the logic and design of the SFS. This avoids the issues with the first option. However, automatic efficient model merging is an open research problem, one that has been demonstrated not fit for software engineering needs [14] nor able to scale [58] beyond polynomial time. The third option is one that we devised as a custom hybrid heuristic whereby we have SimXAMPLE copy the SFS into the SUD (replacing contents), detect any changes to the block signature, and report these to the engineer for their inspection and intervention. This intelligent and user-assisted copy method of SFS application replaces all elements of the SUD with those from the candidate subsystem only after performing a block signature comparison and indicating any differences in the number of inports or outports to the engineer and asking for their intervention. SimXAMPLE realizes the intelligent assisted copy through the application of a heuristic approach that examines the location of the SUD within the MUD as well as the SFS within its respective model file. We represent this heuristic decision process in Fig. 9. The assisted copy method identifies five potential actions, which we place into four scenarios, based on the locations and signatures of the SUD and SFS:

1. **Replace entire model file**: if both the SUD and SFS are top level systems, SimXAMPLE replaces the entire MUD with the model containing the SFS
2. **Copy all contents of SFS into MUD**: if the SUD is a top level system but the SFS is contained within a subsystem, SimXAMPLE places the contents of the SFS at the top level of the MUD without containment
3. **Replace SUD with SFS and connect lines**: if the SUD is contained within a subsystem and connected to the surrounding MUD then the connections become relevant. In this situation SimXAMPLE replaces the SUD with the SFS and connects all ports if and only if there are exactly 1 inport and 1 outport in both the SUD and SFS - *Note:* if the SFS is not contained within a subsystem, SimXAMPLE must first wrap/place it within one before applying this option or the next
4. **Engineer Guided Replacement**: Similar to the previous option, if the SUD is contained, the replacement of the SUD with the SFS is possible.

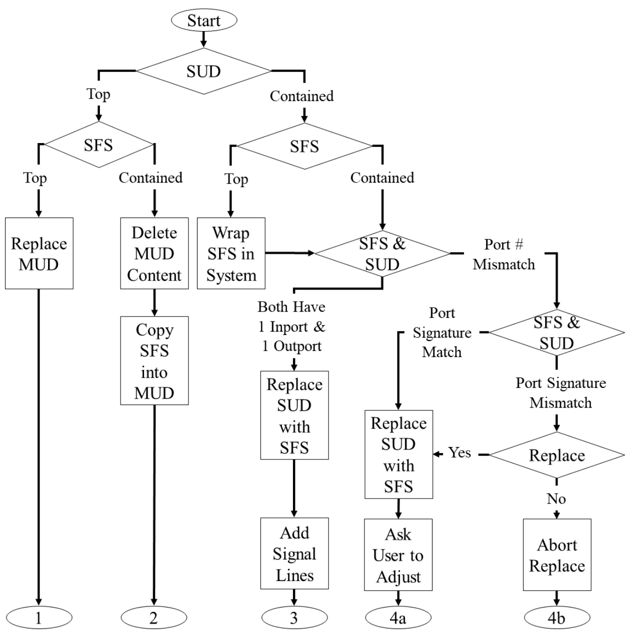
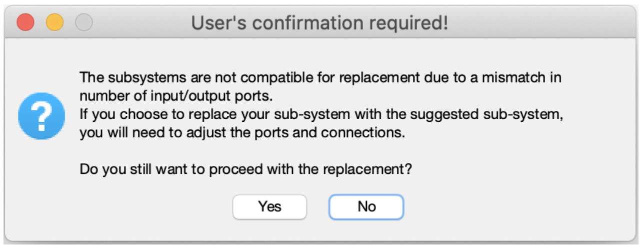


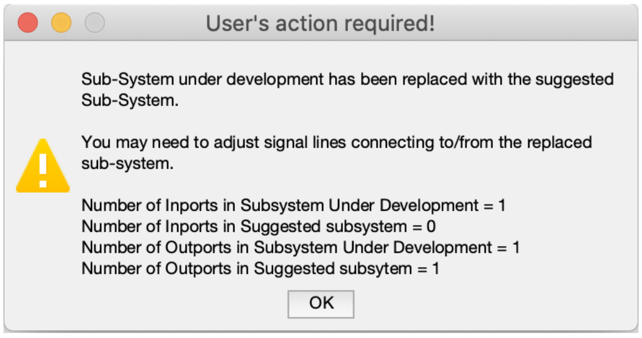
Fig. 9: Decision Process for Inserting SFS into SUD


User's confirmation required!

The subsystems are not compatible for replacement due to a mismatch in number of input/output ports.
If you choose to replace your sub-system with the suggested sub-system, you will need to adjust the ports and connections.

Do you still want to proceed with the replacement?

(a) Prompt to Proceed with Replacement


User's action required!

Sub-System under development has been replaced with the suggested Sub-System.

You may need to adjust signal lines connecting to/from the replaced sub-system.

Number of Inports in Subsystem Under Development = 1
Number of Inports in Suggested subsystem = 0
Number of Outports in Subsystem Under Development = 1
Number of Outports in Suggested subsytem = 1

(b) Notification to Adjust Connections Manually

Fig. 10: Engineer-Guided Replacement Notifications

However, if the block signatures do not have exactly 1 inport and 1 outport, this necessitates engineer notification and intervention, which SimXAMPLE treats in one of two ways:

(a) **Abort Replacement**: if the comparison of the SUD and SFS reveals that either the number of inports do not match or the number of outports do not match, SimXAMPLE asks the engineer if they wish to proceed with replacement given the mismatch (Fig. 10a). If they opt not to, SimXAMPLE aborts the replacement

(b) **Replace SUD with SFS and ask engineer to adjust signal connections**: if the comparison yields a match for both inports and outports, or if the engineer opts to continue merging, SimXAMPLE can make the substitution directly and prompts the engineer to inspect/adjust the connections (Fig. 10b)

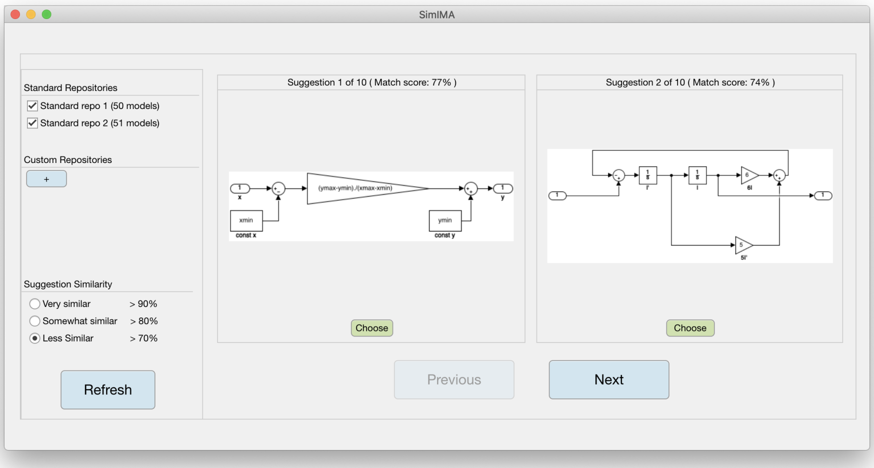
Unless the context and choices lead to aborting the replacement, the suggestion application process can be generalized as follows:

1. compare block-port signatures of SUD and selected SFS
2. prompt user as appropriate (Fig. 10a)
3. remove all existing blocks within SUD
4. copy all blocks from selected SFS into SUD
5. connect SUD to blocks one level above, as appropriate/possible, notifying the user of the need to make additional connections (Fig. 10b)

A successful insertion of the SFS into the SUD constitutes a complete application of SimXAMPLE. A demonstration of this application can be seen in our running example in Section 5.5. SimXAMPLE makes changes primarily within its current modeling context, allowing the engineer to observe any updates on the screen without navigating through the model. The only potential changes made outside of the local context relate to connections to the SUD at a higher level of the model hierarchy, which SimXAMPLE addresses through its user prompts. Following this final phase, the engineer can continue the development of their system, including using SimXAMPLE again for any in-progress or recently completed systems.

5.5 ExampleSUD

ExampleSUD is in a mostly complete state after successive applications of SimGESTION, as we demonstrated in Fig. 6b. We now apply SimXAMPLE to find similar complete systems as suggestions for insertion into the system under development. Querying SimXAMPLE with the ExampleSUD as it exists in Fig. 6b leads to the the suggestions we show in Fig. 11. It is worth noting that the suggestions may appear visually different from the engineer's input model, such as they do in this example. Since Simone's process relies on the underlying textual representations, the similarity values are also based on the textual similarities, which is what



Standard Repositories
☑ Standard repo 1 (50 models)
☑ Standard repo 2 (51 models)

Custom Repositories

Suggestion Similarity
○ Very similar > 90%
○ Somewhat similar > 80%
● Less Similar > 70%

Suggestion 1 of 10 (Match score: 77%)

Suggestion 2 of 10 (Match score: 74%)

Fig. 11: Top two suggestions presented by SimXAMPLE as candidates for insertion into ExampleSUD

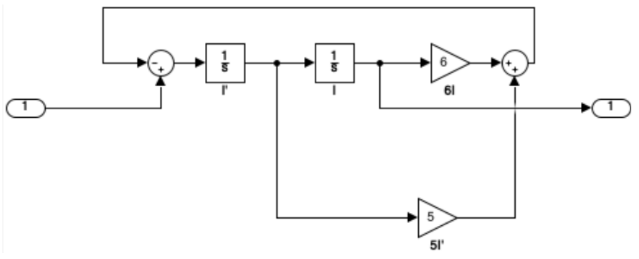


Fig. 12: ExampleSUD after inserting the suggestion from SimXAMPLE

yields the high similarity values, meaning the suggested models, while potentially appearing different, are more similar than one may suspect upon initial visual inspection. Ultimately the engineer is free to choose, or opt not to choose, any of the provided suggestions, or simply use them as inspiration to complete their model. In our running example, we opt to select Suggestion 2 for insertion. After the insertion, ExampleSUD is now complete, as we illustrate in Fig. 12.

6 Evaluation

In this section, we present our evaluation of SimIMA. We evaluate SimGESTION and SimXAMPLE independently, focusing mostly on the evaluation of the approaches/algorithms. At this time, we consider user evaluations and studies out of scope and future work. To be meaningful, such endeavours will require multiple users from different domains. We also omit a formal systematic performance analysis from this current evaluation, as this research was a proof of concept and exploratory, and there does not currently exist a benchmark nor are there comparable tools to measure against. It is our goal, once this framework is fully realized, to provide benchmark data for future approaches to measure against. We do perform brief analyses of performance throughout, but mainly as an internal metric. Our goal in this work is to demonstrate the feasibility of our two forms of modeling assistance and develop a prototype. Thus, we focus our evaluations accordingly. We do additionally use existing work on evaluating IMAs to conduct a qualitative evaluation. We begin with a discussion of the data we use throughout our evaluations.

6.1 Data

For our experiments, we leveraged a large publicly available and curated corpus of Simulink models [22]. Additionally, Boll et al. independently evaluated and validated this set for empirical research [16]. Through investigation and consultation with industry partners they note that the models are suitable for empirical research. Many of the models are "mature" and large enough for analysis purposes. They are diverse enough that they facilitate good replication opportunities as well. Some concerns regarding the set are that some projects are no longer under development and code generation is not well represented. Neither of these are concerns for our purposes, however.

There are a total of 946 model files in this set, including both Simulink MDL (573) and SLX (373) formats. We sorted these models into similar sets based on their domains to provide the most valuable

Table 2: Dataset Information

Domain	**Subsystems**	**Block Types**
Automotive	1258	83
Avionics	2437	72
Electronics	19133	124
Energy	8029	100
Robotics	1090	91
Other	146	51
ALL DOMAINS	32093	130

suggestions. To achieve this we cross referenced the domains indicated explicitly by the corpus curators with our independent analysis of model and documentation content. This left us with 6 non-mutually-exclusive collections: automotive, avionics, electronics, energy, robotics, and other. We include these models, sorted by domain and publish them on a public and persistent repository [1]. They can all also be found either directly on the corpus' website or through a link on that same website. While we could have chosen a variety of models for evaluation, we use these from the corpus for two reasons: to facilitate independent verification and replication of our experiments due to their public availability, and the level of variety and realism in the models as established by the independent evaluation.

To illustrate the diversity of the models used in this analysis, we performed an analysis of the unique block types found in each domain to illustrate the variability of the inputs. A summary of these results can be found in Table 2 along with the number of subsystems contained within the dataset.

6.2 SimGESTION Evaluation

To conduct the evaluation of SimGESTION, we use the established method for measuring prediction accuracy and error classification, K-fold cross validation [9, 33]. This involves splitting data into k complimentary subsets, using some for learning/training and the others for testing/evaluating performance [9], and repeat the process so that each of the k sets is used once as the test set. Specifically, we use 10-Fold Nested Cross-Validation (CV) approach. The outer folds serve to evaluate the prediction models while the inner folds serve to tune/optimize the model hyperparameters. For each category of Simulink models, we first split the entire dataset into 10 folds/parts. Then, we run the evaluation experiments 10 times such that 9 of the folds are used to train the block-prediction models (model-building set), while the remaining 1 fold is used to test the prediction models (testing set). For each of these outer 10 folds, and for each combination of model hyperparameters, the model-building dataset is further split into 10 inner folds. 9 of these inner folds are used to train the prediction models (training set) while the remaining 1 fold (validation set) is used to evaluate their performance. We repeat this process such that each of the inner folds is used as the validation set exactly once, and track the best model hyperparameters throughout he process. We combine this nested cross validation with mutation testing [8] to conduct our evaluation.

6.2.1 Data Preparation

The corpus contains a total of 946 Simulink model files. We discovered 16 of these models from the corpus were invalid because they failed to load successfully in the Simulink versions (R2019b and R2021a) on which we ran these experiments, and removed them immediately. 35 of the model files were not fully navigable, that is, Simulink had errors when trying to fetch "params" for some blocks. So we removed those. Similarly, 362 of these model files were found to be immutable and thus unusable for our purposes. This was either because they had no suitable block to delete at all, or because mutating them would result in a (sub)system with no blocks available to be set as the MRCB. Following the removal of these models, our final model set for SimGESTION consisted of 533 unique Simulink models for our evaluation experiments. Based on our independent analysis of model and documentation content and cross-referencing with the corpus data, we categorized these 533 Simulink models into six categories: automotive, avionics, electronics, energy, robotics, and other, containing 63, 70, 320, 126, 157, and 69 Simulink models respectively. We removed a few randomly selected models from some of these categories to end up with multiples of 10 to better prepare for 10-fold cross validation and a simplified analysis. After this removal, we had the model counts of 60 automotive, 70 avionics, 320 electronics, 120 energy, 150 robotics, and 60 other. Finally, we split each of these categories' model files randomly (using a seed for reproducibility) into 10 folds to for evaluation using 10-fold cross-validation. The decision to separate the model set into domains rather than combining them for a larger set of models to train on was made to ensure that suggestions are being drawn from only models with similar applications. While increasing the data provided to our classifier gives a larger set to train on and classify against, the models from other domains may contain design patterns and elements not typical of the domain of the query model, thus providing distracting suggestions. Considering the expected use case for SimGESTION, the model repositories used for learning would likely be populated with other models from the same domain/industry/organization, so our approach aims to emulate this as closely as possible.

6.2.2 Determining Prediction Classifiers Hyperparameters

SimGESTION has the following hyperparameters,

– ARM has one hyperparameter, which is the amount of antecedent reduction (percentage of antecedents required when assessing with a consequent)
– FREQ has two hyperparameters: the weight of the MRCB destination block types and the weight of the neighbours. These must add up to one.
– Our ensemble classifier has two hyperparameters: the weight given to the ARM classifier and the weight given to the FREQ classifier. These must add up to one.

To determine these values we employed Grid Search [36], the standard method for optimizing hyperparameters. For our FREQ and ensemble hyperparameters, we tuned these by treating their two hyperparameters as one parameter by leveraging the fact that the second hyperparameter is simply 1.0 minus the first hyperparameter. For example, the weight of the neighbours in FREQ is simply 1.0 minus the weight of the MRCB weight. To find the best hyperparameters in each instance, we search through each value of the parameter space, [0,1], using a step size of 0.1 and see the impact on evaluation metric Mean Reciprocal Rank (MRR) using 10-fold cross validation. We arrived at the following configuration hyperparameter values,

– **ARM:** 0.87 antecedent reduction
– **FREQ:** 0.8 MRCB, 0.2 neighbours
– **Ensemble:** 0.44 ARM, 0.56 FREQ

6.2.3 Evaluation of Ensemble Block Suggestion

To evaluate the effectiveness of the ensemble learning method for block suggestions, we apply SimGESTION once for every mutable block within the evaluation set of models. For each block, we delete the block from the model and query SimGESTION for suggestions, with the expected result being the deleted block appearing as a suggestion. This method of mutation creates several different scenarios, for example, where the missing block may be at the beginning of a chain of blocks, the end of the chain, or somewhere in the middle, and with the third scenario being the most commonly occurring. Since the development of Simulink models is often nonlinear, each of these scenarios are important to substantiate our evaluation. In particular, since often development of a subsystem occurs to meet the needs of both its provided inputs and required outputs, it is often the case that subsystems are developed from both the left and right hand sides towards the middle, which matches the most commonly occurring scenario in our evaluation setup. Since the number of suggestions shown in our UI is customizable, we first needed to determine the optimal number of blocks to display to the user, which also formed part of our success criteria. To establish the optimal configuration, we performed an evaluation considering the top-N suggestions, where N ranged from 1 to 10, and measured the prediction accuracy. We plot this comparison in Fig. 13. Based on these results, we determined that displaying the top 6 suggestions provides strong accuracy without overcrowding the engineer's screen with suggestions. While increasing the number of suggestions presented to engineers marginally increases the accuracy, the gains demonstrated are not significant enough to increase the default number of suggestions. Furthermore, should the engineer wish, they are able to increase the number of suggestions provided to them through the SimGESTION Configuration Wizard presented in Section 4.2.

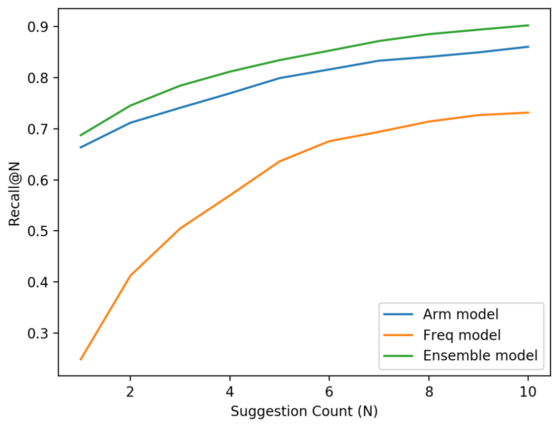


Fig. 13: Accuracy of Suggestion Based on Number of Suggestions Shown

Based on this optimal configuration, we ran our validation experiments with the constraint that the deleted block must be suggested as one of the top six suggestions. If the deleted block appears in the top six suggestions, SimGESTION has made a correct suggestion, and if the correct block does not appear, or was ranked below the sixth suggestion, SimGESTION did not provide the correct suggestion.

We chose the single-block mutation-testing style of evaluation as the most direct method of evaluating our suggestions as it allowed us to automatically identify whether the correct suggestion was made without the need for a domain expert. Since the deleted block is, by definition, the correct suggestion for the given query environment, its presence among the suggested blocks provides a clear indication of prediction correctness. Since it is not guaranteed the correct block is suggested in the top position, an experiment where there were multiple mutations applied would not have been feasible. In that design, if we opted to select the top block for insertion after each query, and the correct block was not ranked highest, any subsequent queries may produce suggestions further away from the original model since an incorrect context was applied. While our approach may be seen as an optimistic experimental design, it provides the only way of guaranteeing correctness beyond external validation, which is outside the scope of this experiment.

6.2.4 Results

We evaluated each set of models independently to ensure that the block-prediction models were trained on models from a similar domain to provide the most effective suggestions. For each domain, we identify the Mean Reciprocal Rank (MRR) and Recall@6 of suggestions produced by all three prediction models. MRR was chosen to accurately represent the average precision of suggestions by taking the mean of the reciprocal of the rank of the first correct suggestion for each query, and includes a penalty if the correct result is not returned [72]. This is shown in Equation 1, where Q represents the number of queries and $rank_i$ is the position of the correct suggestion for that query. For example, if over four suggestions the correct result was found in positions 1, 1, 2, and 4, they would receive reciprocal ranks of 1, 1, 1/2, and 1/4, with an MRR of 0.6875. Recall@6 is defined in Equation 2 where TP represents the number of times the correct result is found in the top six suggestions (a true positive) and FN represents the times where the correct result is not presented in the top six suggestions (a false negative). For example, if in ten queries, the correct result is found in the top six suggestions nine times, and one time it is not in the top six, the Recall@6 would be 0.9.

$$MRR = \frac{1}{Q} \sum_{i=1}^{Q} \frac{1}{rank_i} \tag{1}$$

$$Recall@6 = \frac{TP}{TP + FN} \tag{2}$$

Table 3: SimGESTION Suggestion MRR and Recall

Prediction Model	**Domain**	**MRR**	**Recall@6**
Arm	automotive	0.6	0.65
Arm	avionics	0.53	0.85
Arm	electronics	0.65	0.81
Arm	energy	0.63	0.85
Arm	robotics	0.59	0.7
Arm	other	0.54	0.62
Arm	**ALL_DOMAINS**	**0.59**	**0.82**
Freq	automotive	0.41	0.56
Freq	avionics	0.4	0.83
Freq	electronics	0.39	0.67
Freq	energy	0.38	0.67
Freq	robotics	0.47	0.7
Freq	other	0.46	0.74
Freq	**ALL_DOMAINS**	**0.42**	**0.68**
Ensemble	automotive	0.63	0.74
Ensemble	avionics	0.63	0.91
Ensemble	electronics	0.66	0.85
Ensemble	energy	0.68	0.87
Ensemble	robotics	0.68	0.8
Ensemble	other	0.68	0.74
Ensemble	**ALL_DOMAINS**	**0.66**	**0.85**

We present the detailed results in Table 3. We also provide MRR and recall values for each prediction model averaged over all domains to indicate the performance of the prediction models in general.

6.2.5 Discussion

Regarding our first research question, we have demonstrated that we can provide engineers with step-wise suggestions that are based on the analysis of a configurable model set. More specifically, we see that, for the majority of domains, our accuracy is in the low 80s. A notable outlier is our experiments with the avionics dataset, which also impacts our total accuracy score significantly. To elaborate, both individual prediction models, as well as their ensemble, have low prediction accuracy on the Avionics dataset, especially compared to the other domains. One possible explanation of this observation is that the Simulink models within the Avionics dataset are not as mutually similar as models in other data sets. As a result, for this dataset, the prediction models are trained poorly and hence make inaccurate predictions.

6.3 SimXAMPLE Evaluation

To evaluate our research and corresponding application, we conducted experiments in pursuit of two avenues of evaluation: evaluation of our visualization of the inferred suggestions to engineers (which items are shown / filtered) consistent with how code recommender systems are evaluated [26] and evaluation of the application of those suggestions to the engineers' interfaces. While a scenario based evaluation where a user assembles a model independently and then queries SimXAMPLE for suggestions may provide interesting results, the inability to meaningfully automate this on a large scale and the current lack of a comparable benchmark leaves little opportunity for a systematic evaluation of this type at this time. Instead, we focus on the visualization and insertion of example systems into the models under development.

6.3.1 Data Preparation

We discovered 16 of the MDL-formatted models from the corpus were invalid, and removed them immediately. To facilitate Simone clone detection, we first converted the SLX models to the MDL format required by Simone using Matlab's conversion process. Five of the SLX-formatted models failed to convert to MDL format due to unsupported characters. After conversion, we had a total of 925 MDL-formatted models (557 original + 368 converted). Through further analysis, we discovered that 259 of these models were immutable, thus incapable of being mutated for our experiments. This gave us a total of 666 valid, mutable, MDL-formatted Simulink models. Of these models, 414 models were incompatible with the latest Simone MDL grammar. In particular, we discarded the entire "Electronics" dataset as almost all models in that set were incompatible with Simone. Following the removal of these models, our final test set consisted of 252 unique MDL models for our evaluation experiments.

6.3.2 Evaluation of the Visualization of Inferred Suggestions

To validate correctness of the inferred suggestion visualization, we employ mutation testing, a methodology that has been demonstrably successful in testing experiments [8], including clone-related experiments [57, 34]. Specifically, we determine if SimXAMPLE recommends the completed version of a mutated model as one of its top suggestions. This is consistent with recommender evaluation strategies that consider rank and retrieval [26]. It is important to stress that we are not conducting an evaluation of Simone's clone detection results as this has been accomplished in the past in a systematic evaluation of model clone detection tools [70]. Instead, our evaluation demonstrates our approach's configuration of Simone for our purposes and our interpretation and visualization of its resulting data. Through this evaluation we aim to illustrate that SimXAMPLE uses the appropriate parameters and applies the correct sorting and filtering of Simone results to visualize optimal and appropriate model examples in a novel and useful manner.

To conduct this experiment, we mutate each of our test models using existing established Simulink mutations [68] at random subsystems within the models. We delete a single randomly-selected block from each SUD to replicate an incomplete subsystem for which SimXAMPLE can provide suggestions. We scripted this mutation using a seeded randomization such that the process could easily be reproduced. Multiple/compound mutations would cause significant distance to the point of no longer being a clone, and thus are not relevant in our context. Due to the varying size of SUDs, deleting more than a single block could lead to significant differences, making the unmutated model undetectable through clone detection, which is why a single-mutation method was employed. This application of mutation is optimal for testing the inferred suggestion as we are concerned primarily with retrieving the closest match from the repository, which, by design, will be the unmutated model. Increasingly mutated models would yield a larger difference from the intended replacement, which would still be included in the result, however, possibly in a lower ranked suggestion. All our mutations and random seeds are in our public repository for replication purposes.

We used each of the 252 mutated models as MUDs and invoked SimXAMPLE each time with all models in the set used as the repository of potential clones to provide the maximum variability of models in the repository. For SimXAMPLE to provide a suggestion correctly and successfully, we decided that the completed subsystem from the original model must be within the top 2 suggestions provided by SimXAMPLE, as this means the option is visible and available without engineers having to browse suggestions. While the ideal result is to have the model that we mutated be the top suggestion in each application, the similarity of some of the subsystems in our test set allows for a variety of suitable suggestions. For each input, we observed the rank of the "correct" subsystem presented by SimXAMPLE and noted whether a correct suggestion was made. The intent of our evaluation is to ensure that our approach interprets the clone detection results to return and visualize the appropriate recommendations.

6.3.3 Evaluation of the Replacement

Our evaluation of the replacement focuses on the correctness of SimXAMPLE's insertion of the SFS into the SUD. We define a correct insertion as one in which the resulting SUD must contain exactly the elements from the SFS, with appropriate connections, and with notifications being displayed to the engineer when SimXAMPLE cannot make connections automatically.

To realize this evaluation, we simplified the clone detection and suggestion portions of the process, as we accomplished this in the first part of our evaluation independently. We did not find a single model in our model set which we could use directly as the Model Under Development (MUD) to test all five possible scenarios from Fig. 9. Therefore, we created a custom Simulink model based on the Matlab Central model *Rotfe25x/Rotfe25x/models/sim_tutorial.mdl* as it required only a deletion of 1 specific block to allow us to test each of the five possible cases. To prepare the repository set for this second experiment, we modified the same base-model deliberately to produce two different Simulink models. Each of these models contained clones of the MUD at various levels of containment. Thus, our custom MUD, together with the custom repository allowed us to run SimXAMPLE in different locations within the experimental model to realize the 5 replacement options. We executed SimXAMPLE following the test scenarios in Table 4 and observed the outcomes against the expected outcomes in that same table. This evaluation experiment served as a correctness test to ensure proper insertion of the SFS into the SUD. While only one input model was used, we demonstrate full coverage of all five scenarios through its invocation in various model locations and by responding to the prompts in the correct way to yield the five specific results. This method of evaluation ensures that all possible insertion scenarios are evaluated with minimal noise introduced through unnecessary modifications to the source models.

6.3.4 Results

We first present the results of our validity experiments on inferred suggestions in Table 5. For each of the five model sets, we present the number of models where clone detection was successful, and of those, the number of mutated models where SimXAMPLE found the original/unmutated corresponding model as a clone. Furthermore, we narrow this number to indicate the instances where the correct clone is found in position 1 or 2, indicating instances where the engineer is not required to browse to find the correct suggestion, as it has been visualized on the initial results page.

We present the results of our evaluation of the insertion of the SFS through our experiment targeting the five potential insertion scenarios in Table 4. We compared each of the five resulting events against the expected results to determine its success. Of the five test scenarios, SimXAMPLE demonstrated correct insertion behavior in all five cases. We include all necessary data and methods necessary for replication and verification in our public repository [1].

6.3.5 Discussion

Our second research question has been answered. We have devised an approach that correctly provides engineers with similar model examples based on analysis of a configurable model set. When it comes to our specific results, since the data sets are not mutually exclusive, calculating accuracy based on their totals is not possible. Through our evaluation of the suggestions identified and visualized by SimXAMPLE, we demonstrated 82.05% accuracy with our test set across all 5 domains. It is also worth calculating the accuracy after excluding the models in the “other” set. The reason is clone detection, and thus SimXAMPLE, is most useful when compared to similar sets/domains. After omitting this category, SimXAMPLE’s average accuracy increases to 85.23%. If we consider any time SimXAMPLE finds the correct clone, rather than in the first or second position only, the accuracy is increased to 90.83% overall and 92.25% when excluding the “other” category. This accuracy is not necessarily the case in all contexts, but was employed by us for evaluation to confirm SimXAMPLE’s ability to interpret and process the model clone detection results to suggest the most appropriate subsystems.

Since we based correctness on finding the original/unmutated model as a top-two suggestion, we were expecting 100% accuracy given a single block deletion and the use of a 30% difference threshold for clone detection. However, it is possible that the block-deletion mutation causes an over mutation such that Simone is unable to find the correct suggestion. This, however, is a shortcoming of Simone’s sensitivity rather than SimXAMPLE’s application, interpretation, and visualization of the results. SimXAMPLE always displays the correct suggestion indicated by the Simone clone report.

In regard to performance, we observed in our experiments that as the model repositories increased in size, the time required to perform clone detection increased as well. Thus, we recommend selecting only the most relevant and necessary models as repositories. Regarding the evaluation of the insertion, we designed the experiment to demonstrate coverage of the five possible outcomes of insertion. Since the act of insertion takes the form of a complete copy and replacement of

Table 4: Evaluation Tests for the 5 Insertion Cases

	Scenario	**Expected Result**
1	delete block at top level of MUD, choose suggestion of large top level model	Entire MUD replaced with entire model of SFS
2	delete block at top level of MUD, choose suggestion of the contained full system	Entire MUD replaced with contents of SFS
3	delete block within subsystem with 1 inport and 1 outport, choose suggestion with matching port signature	SUD replaced with SFS, lines automatically connected
4a	delete block within subsystem, choose non-exact suggestion, opt to proceed	SUD replaced with SFS, prompted to manually adjust connections
4b	delete block within subsystem, choose non-exact suggestion, opt to abort	Replacement Aborted

Table 5: Inferred Suggestion Accuracy

Domain	**Models in set**	**Correct clone presented (Recall@6)**	**Clone ranked in position 1 or 2 (Recall@2)**
automotive	63	59 (93.7%)	56 (88.9%)
avionics	69	66 (95.7%)	63 (91.3%)
energy	53	49 (92.5%)	45 (84.9%)
robotics	47	41 (87.2%)	38 (80.9%)
other	101	86 (85.2%)	63 (62.4%)

the SFS into the SUD using atomic Matlab commands, this two-step operation completes in constant time.

The success of this first time realization of an intelligent model assistant using model clones is indicative of significant positive impacts of virtual intelligent modeling assistance. Through reasonably accurate suggestion, visualization, and insertion of candidate subsystems, our SimXAMPLE process may assist modelers in becoming more familiar with standard design principles and effective modeling practices by example. The well-documented gains associated with code completion implementations [54] are likely to transfer to users of modeling assistants such as SimXAMPLE. Furthermore, our realization and visualization of suggestions through the use of model clone detection has significance on its own. It provides a new application for existing model clone detection tools and an accompanying novel visualization. While our approach employs Simone due to our earlier stated reasons, the modular nature of our process allows for any model clone detector to be used by tailoring Phase 1 to a respective detector.

6.4 Qualitative Evaluation via IMA Assessment Grid

Mussbacher et al. describe their initial ideas performing assessment of intelligent modeling assistance [45]. While they consider it only a first step, we believe it prudent to apply their assessment grid to SimIMA as appropriate. Specifically, two of the authors independently self assessed SimIMA along that grid and came together to reconcile final scores. We summarize our scores in Table 6 and direct the reader to Mussbacher et al.'s work for more details on the analysis criteria. In regard to Quality of the IMA regarding models, we gave SimIMA a score of 1 - Syntactic Quality as syntactic quality is enforced and always produced by SimXAMPLE and SimGESTION. We could not give a score of 2 nor 3, as it depends on how the modeler chooses both to configure the repositories and employ the suggestions. Regarding Autonomy, SimIMA scores Level 2 - Narrowed Set. This is because SimIMA provides a list of multiple suggestions that have been narrowed down. When it comes to Relevance, the assessment grid considers accuracy as an acceptable metric. For SimGESTION, our relevance is Level 79. For SimXAMPLE, our relevance is 82. The Confidence property from the grid for SimIMA is 100 as we always (100%) show a confidence value. Our Trust level for SimIMA is 0 as SimIMA gathers no information about the modeler's trust. We leave such feedback as future work. Explainability for SimIMA is also Level 0 currently as we do not provide any insight on how its insight was gathered. For the Quality Degree of SimIMA, we evaluated it at a Level 2 - Safe Lookup. This is because we consider the context and external sources. While we do not leverage syntactic or semantic descriptions, we do allow the engineer to tailor their predictive models based on tradeoffs. SimIMA has a Timeliness of Level 2 - Short running and regularly (out of a possible level 5). SimGESTION has FREQ complete suggestions around roughly one second, and its retrieval time is independent of the training data size. For ARM, it also computes suggestions around one second, however, its retrieval time does increase with training data size. SimXAMPLE also takes a matter of seconds. It is not level 3 as it is not iterative, nor level 4 as it is not less than one second in running time.

Regarding the Quality of SimIMA's External Sources, we gave it Levels S1 - Project and infrastructure, A2 - Public, U3 - Periodically updated, and C2 - Curated. For S1, SimIMA uses the current MUD and SUD, and past (repositories) development artifacts. It also uses data, information, and knowledge about Simulink. We gave ourselves Level A2 as all repositories included with SimIMA are public, as are the ones from our experiments. We deemed Level U3 appropriate for SimIMA as we can easily update the standard repositories that we included with SimIMA regularly. Lastly, Level C2 seemed most appropriate since the repositories we include with SimIMA and used in our experiments are from a curated dataset.

6.5 Threats to Validity

6.5.1 External Threats

An external threat to validity lies in our choice of Simulink to research and demonstrate our original vision [67]. Our vision is language agnostic, and nothing in the way we defined and realized our approach in this work is strongly language specific. That is, while our research prototype exploits certain characteristics and features of Simulink, our general process is designed to be applicable and generalizable to any modeling language. While our initial research provides a proof of concept and an implementation using Simulink models, the concepts related to repository mining for stepwise suggestions and clone detection for full model completion should apply for most graphical modeling languages. Migration to other language frameworks may require significant implementation efforts, however, the general principles demonstrated with SimIMA should reasonably apply in any context.

For SimXAMPLE, this is true for any language in which Type 3 model clone detection exists. Simulink features the most prevalent and mature Type 3 clone detection, which is one of the reasons we chose it. Until our general process is applied to another modeling language, a threat to validity is its generality. However, more Type 3 model clone detectors need to be developed [69]. Similarly, our choice of Simone for SimXAMPLE represents a threat to external validity. While we motivate and justify choosing Simone due to its maturity and evaluation history, it is a threat that we did not experiment with other clone detectors instead of Simone.

An additional external threat to validity is our selection of models in our experiments. While we already discussed our motivation for choosing this set; including it being a large, publicly-available, and curated set; using industrial models would reduce this threat, at the cost of public availability and replicability. Related to this, the use of existing models within our data sets assumes some level of correctness or applicability of the models, which cannot always be guaranteed. While there is some onus on the the engineer to only use relevant and stable models, it is possible through the inclusion of other models, the suggestions may not be as expected.

We also have the external threat of the experimenter effect. Specifically, in us having members of our research team doing the experiments. For the quantitative evaluations, we had a different member of our team chose the evaluation model set than the team members who actually developed SimIMA. However, the actual experiments were conducted by the same members that created SimIMA. Experimenter effect is even more of a concern in our qualitative evaluation where we had two members of our team perform self assessment. We acknowledge that originally in that discussion, and leave third-party assessment through the assessment grid as future work.

6.5.2 Internal Threats

Currently, the suggestions provided by SimGESTION are at the block level, with default configurations and values used when they are inserted. This approach provides minimal diversity of suggestions, which leads to potentially naive models without further user intervention. While a shortcoming, this approach is quite similar to many current code completion implementations, which provide paremeterized functional calls with default parameters rather than specific values, thus should not be considered a significant limitation. While there are some exceptions through smart code completion [35,13,61,51], our approach is analogous to most standard code completion implementations.

Additionally, in SimGESTION our ensemble approach relies more on block presence than it does on temporality. While block presence in a semi-automated (suggestion-based) use case such as our is very useful, temporal aspects can also provide important information, especially in a language such as Simulink. However, as this was a proof of concept of a new approach, we leave temporaility as future work.

The selection and use of both ARM and FREQ were based on our analysis of available techniques combined with the data available from the model repository. Since no formal comparison of alternative techniques was conducted, it is possible that some other classification/suggestion algorithms may have yielded better performance.

Table 6: Summary of Qualitative Assessment

Property	SimIMA Score
Quality of IMA Regarding Models	Level 1 - Syntactic Quality
Autonomy	Level 2 - Narrowed Set
Relevance	Levels 79 and 82
Confidence	Level 100
Trust	Level 0 – Unidentified
Explainability	Level 0 – No explanation
Quality Degree	Level 2 - Safe Lookup
Timeliness	Level 2 – Short running and regularly
Quality of IMA Regarding External Sources	Level S1 - Project and infrastructure Level A2 - Public Level U3 - Periodically updated Level C2 - Curated

Without conducting a participant evaluation with Simulink modelers, we are unable to fully evaluate the correctness, usefulness, and applicability of the suggestions provided by SimIMA and it's two processes from a user perspective. While all of our internal evaluations indicate positive results for the suggestions, further evaluation would have strengthened these results. Unfortunately, the conduct of this type of user study was found to be too difficult to design and implement, and recruiting suitable participants was beyond our current capabilities.

7 Related Work

We compare our work to existing research in model assistance and completion. Research that performs the same functions and services as SimIMA specifically is relatively sparse in comparison to its source code analog. There is related modeling research that has goals that align with ours, and most importantly, achieves those goals through means that are similar and related. We include both in this section.

Dyck et al. outlines guidelines for how UML model recommendation systems can be implemented based on a survey of UML modeling tools in practice [29]. Their requirements and suggested architectures were very helpful and inspirational in the design of SimIMA and its integration into the Simulink.

Using XML and RDF, Segura et al. devised a meta-model modeling language creation assistant [63]. The MetaModelAgent tool extension in Eclipse works similarly [4]. Meta-model language development differs from our target of model creation. Mazanek et al. provide model editors assistance by considering general graph grammars and generating suggestions for incomplete graphs of models [44]. Sen et al. help model editors complete models automatically by evaluating and interpreting the modeling language meta model [64]. Similarly, Steimann and Ulke use a combination of a modeling language analysis and constraint solving to compute suggest potential model modifications [65]. DoMoBOT also has a focus on providing recommendations using natural language [60]. Specifically, it generates a domain model using natural language processing and attempts to automatically modify/update the domain model accordingly. Proactive modeling is another example that uses constraints, along with DSML semantics, to try and predict model edits to assist modelers [48]. Recently, Nair et al. extended the Proactive Modeling Engine to provide recommendations using Object Constraint Language constraints and past modeling actions and history [47]. Burgueno et al. also employ natural language processing to devise an assistant that provides completion suggestions for completion of domain models [20]. Kuschke et al. realize completion suggestions based on UML activity design patterns [42,41]. While, they use histories in the form of model operations to derive completion suggestions, our work finds examples for completion by structurally similar model clones. Additionally, their suggestions are based exclusively on predefined modeling activities for structural UML models, while ours is based on an analysis of similar models, not activities. All of these related works differ from SimIMA in both the type of analysis and knowledge base considered by their respective assistants. SimIMA looks for suggestions derived from existing models in repositories specified by the engineers, such as those from the same domain, same project/organization, and exemplars. Di Rocco et al. present MemoRec, a similar approach to model recommender systems that uses collaborative filtering to recommend entities related to metamodel under construction [27], which employs similar techniques related to similarity analysis between exising models and the current model, but applied to a different modeling environment. Weyssow et al. similarly approach intelligent modeling assistance through the creation of metamodel recommendations. They employ a similar data-driven approach like SimIMA, however they leverage both lexical and structural properties of the metamodels [73], whereas SimIMA focuses on patterns of occurrence.

There is a related feature to SimXAMPLE included in Matlab Simulink, which was released in the 2018a version. They use proprietary clone detection to allow Simulink developers to refactor models through library replacement[5]. While both SimXAMPLE and their approach use model clone detection, their approach is intended explicitly to refactor Simulink models that are *finished and complete*. In contrast, SimXAMPLE is tailored for engineers to create and develop/edit models that are *incomplete model (fragments)*, a different activity and focus. There are also differences in the two model clone detection approaches. Their model clone detection is private/proprietary and can detect *only* Type 3 near-miss clones that have "different block parameter values". This results in their built-in tool not being able to detect the relatively common [68] occurrence of Type 3 model clones that have/use different types of blocks or those that have additional blocks. While we realized and demonstrated SimXAMPLE through a tool created specifically for Simulink, our research is intended to be applicable to any model type for which there exists model clone detection techniques as we disseminate and generalize our work and artifacts. Similarly to that Matlab feature, endogenous model transformations have been used by Kappel et al. to assist in refactoring existing *completed* models only [38] through transformations. The Extremo plugin accomplishes meta model and model reuse by creating repositories from a variety of data sources, including other models [62]. However, it explicitly 1) is for reuse rather than model completion and editing, 2) does not perform any model comparison/clone detection instead requiring a engineer to query and navigate manually imported data repositories, and 3) is designed to work for tools within the Eclipse environment only. Elkamel et al. perform XMI comparison to find similar UML classes using metrics [32]. Its purpose is similar to SimXAMPLE but it is limited in its ability to find related classes in that it is unable to detect many of the Type 3 clones that SimXAMPLE can as they focus on name similarity of classes, attributes, and operations. SimXAMPLE's use of Type 3 clone detection allows for providing identical, renamed (ignoring names), and near-miss (different structure and missing/added elements) suggestions.

[5] https://www.mathworks.com/help/slcheck/ug/identify-subsystem-clones-and-replace-them-with-library-blocks.html

Barriaga et al. described repairing software models using reinforcement learning [15]. Their work differs from ours both in context and methods. In contrast to our research, they focus specifically on repairing models that are known to be broken with errors that have a concrete list of possible actions. Ours is intended to be used on incomplete model fragments and can also be used on complete models, irrespective of their "brokenness". Their method is centered on rewards associated with actions, which is different than finding similar examples or evaluating existing artifacts.

8 Conclusion

In this paper, we describe our research in determining the plausibility of providing software modeling assistance by using machine learning and model clone detection on past artifacts. To that end, we answered two subquestions targeted to different forms of assistance: providing step-wise (element-level) suggestions and providing similar example models as suggestions. These forms of assistance correspond to two components of our larger SimIMA assistant: SimGESTION and SimXAMPLE. SimGESTION employed ensemble learning by combining association rule mining and frequency information. SimXAMPLE configured, executed, and visualized model clone detection and (sub)system replacement.

To evaluate all of SimIMA, we used a large, publicly available, independently validated, and curated data set to facilitate reproduction and reproducibility. We employed k-cross fold validation on SimGESTION to tune its hyperparameters and also evaluate its results. Doing so allowed us to derive the optimal weight parameters for its various classifiers and its accuracy. In our experiments on various domains, SimGESTION ended up with a total accuracy of around 79%. For SimXAMPLE, we were able to provide the correct/accurate suggestions roughly 82% of the time. While our overall intent was to establish the feasibility model assistants employing machine learning and model clone detection, we believe these accuracy results are very encouraging.

8.1 Future Work

As with most recommender systems, there is always ample opportunity to improve the suggestions provided. One planned enhancement relates to the current shortcoming where SimGESTION provides block suggestions without customizing the block's parameters. This functionality, while significant, was specifically omitted from our initial realization due to the lack of sufficient relevant data within the dataset. Further, our lack of expertise in Simulink modeling at an industry level prevented us from providing meaningful parameter suggestions. We aim to work with professionals proficient in Simulink to better understand the types of parameters and how best to make suggestions for these values, since the methods currently employed for SimGESTION that rely solely on repository data are not likely to produce relevant suggestions. Through additional analysis of models and discussions with domain experts, it is potentially possible for SimGESTION to suggest different values for blocks, in addition to different block types. For example, rather than just suggesting a *Constant* block, SimGESTION could suggest a *Constant* block with a value of 10, or provide possible conditions for an *if* block.

From an evaluation perspective, there are a number of avenues we are considering for future work. Now that we have demonstrated the plausibility of these assistants and developed working prototypes, we want to connect and work with industry to perform user evaluations. To better demonstrate the robustness and effectiveness of SimIMA it would be beneficial to validate using industrial production models using Simulink practitioners in a robust user study to elicit feedback on the correctness of the suggestions and insertions. By having SimIMA applied in a more real-world setting, we would better be able to help foster use and promotion. This is not an indictment of our current evaluation. Rather, application of SimIMA is very context dependent and having industrial use cases would expand the context of our evaluation. Additionally, we want to recruit third party researchers, possibility from the original RF-IMA assessment grid paper, to conduct an independent qualitative evaluation.

As with most research, there is potential for user interface (UI) enhancements. Careful consideration was part of our SimIMA design process, as was adhering to established guidelines for creating model recommendation systems [30]. However, conducting a study with Simulink practitioners to solicit feedback on the UI design and usability would be beneficial. We would be able to consider alternative UI designs and determine which features contribute the most to the usability and design. Even through our qualitative assessment, we identified areas of potential enhancements such as more explanations and building trust through transparency.

Finally, as we propose this work as generalizable to other modeling environments and languages, it will be necessary to explore the application of our machine learning and clone detection based approaches with other modeling languages to better support this claim.

Acknowledgements This material is based upon work supported by the National Science Foundation under Grant No. 1849632.

References

1. Adhikari, B., Rapos, E.J., Stephan, M.: Initial Evaluation Data for SimIMA: A Virtual Simulink Intelligent Modeling Assistant (2021). DOI 10.5281/zenodo.5123565. URL `https://doi.org/10.5281/zenodo.5123564`
2. Adhikari, B., Rapos, E.J., Stephan, M.: Simulink Intelligent Modeling Assistant (SimIMA) (2021). DOI 10.5281/zenodo.5123570. URL `https://doi.org/10.5281/zenodo.5123568`
3. Adhikari, B., Rapos, E.J., Stephan, M.: Simulink model transformation for backwards version compatibility. In: 2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), pp. 427–436 (2021). DOI 10.1109/MODELS-C53483.2021.00066
4. Adocus AB: MetaModelAgent Concept. `http://www.metamodelagent.com/concept.html`. Accessed: 2021-04-01
5. Agrawal, R., Imieliński, T., Swami, A.: Mining association rules between sets of items in large databases. In: Proceedings of the 1993 ACM SIGMOD international conference on Management of data, pp. 207–216 (1993)
6. Alalfi, M.H., Cordy, J.R., Dean, T.R., Stephan, M., Stevenson, A.: Models are code too: Near-miss clone detection for simulink models. In: International Conference on Software Maintenance, pp. 295–304. IEEE, Riva del Garda, Trento, Italy (2012)
7. Almonte, L., Guerra, E., Cantador, I., De Lara, J.: Recommender systems in model-driven engineering. Software and Systems Modeling **21**(1), 249–280 (2022)
8. Andrews, J.H., Briand, L.C., Labiche, Y.: Is mutation an appropriate tool for testing experiments? In: Proceedings of the 27th international conference on Software engineering, pp. 402–411 (2005)
9. Anguita, D., Ghelardoni, L., Ghio, A., Oneto, L., Ridella, S.: The'k'in k-fold cross validation. In: ESANN, pp. 441–446 (2012)
10. Antony, E., Alalfi, M.H., Cordy, J.R.: An Approach to Clone Detection in Behavioural Models. In: International Working Conference in Reverse Engineering, pp. 472–476 (2013)
11. Asaduzzaman, M., Roy, C.K., Schneider, K.A., Hou, D.: Cscc: Simple, efficient, context sensitive code completion. In: International Conference on Software Maintenance and Evolution, pp. 71–80. IEEE, Victoria, BC, Canada (2014)
12. Babur, Ö., Stephan, M.: Mocop: towards a model clone portal. In: 2019 IEEE/ACM 11th International Workshop on Modelling in Software Engineering (MiSE), pp. 78–81. IEEE, IEEE, Montreal, QC, Canada (2019)
13. Barath, B., Knottenbelt, W., Heinis, T.: Improving code completion with machine learning. Imperial College London (2020)
14. Barrett, S., Chalin, P., Butler, G.: Model merging falls short of software engineering needs. In: Proc. of the 2nd Workshop on Model-Driven Software Evolution. Citeseer (2008)
15. Barriga, A., Rutle, A., Heldal, R.: Personalized and automatic model repairing using reinforcement learning. In: 2019 ACM/IEEE 22nd International Conference

on Model Driven Engineering Languages and Systems Companion (MODELS-C), pp. 175–181. IEEE, IEEE, Munich, Germany (2019)

16. Boll, A., Brokhausen, F., Amorim, T., Kehrer, T., Vogelsang, A.: Characteristics, potentials, and limitations of open source simulink projects for empirical research. Software and Systems Modeling **tbd**(tbd), 20pp (2021). In press
17. Brambilla, M., Cabot, J., Wimmer, M.: Model-driven software engineering in practice. Synthesis lectures on software engineering **3**(1), 1–207 (2017)
18. Bruch, M., Monperrus, M., Mezini, M.: Learning from examples to improve code completion systems. In: Joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, pp. 213–222 (2009)
19. Bucchiarone, A., Cabot, J., Paige, R.F., Pierantonio, A.: Grand challenges in model-driven engineering: an analysis of the state of the research. Software and Systems Modeling pp. 1–9 (2020)
20. Burgueño, L., Clarisó, R., Li, S., Gérard, S., Cabot, J.: A NLP-based architecture for the autocompletion of partial domain models (2020). URL `https://hal.archives-ouvertes.fr/hal-03010872`. Working paper or preprint
21. Cabot, J., Clarisó, R., Brambilla, M., Gérard, S.: Cognifying model-driven software engineering. In: Federation of International Conferences on Software Technologies: Applications and Foundations, pp. 154–160. Springer, Springer, Marburg, Germany (2017)
22. Chowdhury, S.A., Varghese, L.S., Mohian, S., Johnson, T.T., Csallner, C.: A curated corpus of simulink models for model-based empirical studies. In: 2018 IEEE/ACM 4th International Workshop on Software Engineering for Smart Cyber-Physical Systems (SEsCPS), pp. 45–48. IEEE (2018)
23. Cordy, J.R.: Submodel pattern extraction for simulink models. In: International Software Product Line Conference, pp. 7–10 (2013)
24. Dean, T.R., Chen, J., Alalfi, M.H.: Clone detection in Matlab Stateflow models. Electronic Communications of the EASST **63** (2014)
25. Deissenboeck, F., Hummel, B., Jürgens, E., Schätz, B., Wagner, S., Girard, J.F., Teuchert, S.: Clone detection in automotive model-based development. In: International Conference on Software Engineering, pp. 603–612 (2008)
26. Del Olmo, F.H., Gaudioso, E.: Evaluation of recommender systems: A new approach. Expert Systems with Applications **35**(3), 790–804 (2008)
27. Di Rocco, J., Di Ruscio, D., Di Sipio, C., Nguyen, P.T., Pierantonio, A.: Memorec: a recommender system for assisting modelers in specifying metamodels. Software and Systems Modeling pp. 1–21 (2022)
28. Dietterich, T.G.: Ensemble methods in machine learning. In: International workshop on multiple classifier systems, pp. 1–15. Springer (2000)
29. Dyck, A., Ganser, A., Lichter, H.: A framework for model recommenders requirements, architecture and tool support. In: International Conference on Model-Driven Engineering and Software Development, pp. 282–290 (2014)
30. Dyck, A., Ganser, A., Lichter, H.: On designing recommenders for graphical domain modeling environments. In: International Conference on Model-Driven Engineering and Software Development, pp. 291–299 (2014)
31. Eclipse Foundation: Code recommenders (2017). `http://www.eclipse.org/recommenders/` accessed on 2017-07-31
32. Elkamel, A., Gzara, M., Ben-Abdallah, H.: An uml class recommender system for software design. In: 2016 IEEE/ACS 13th International Conference of Computer Systems and Applications (AICCSA), pp. 1–8 (2016). DOI 10.1109/AICCSA.2016.7945659
33. Fushiki, T.: Estimation of prediction error by using k-fold cross-validation. Statistics and Computing **21**(2), 137–146 (2011)
34. Gautam, P., Saini, H.: Mutation testing-based evaluation framework for evaluating software clone detection tools. In: Reliability and Risk Assessment in Engineering, pp. 21–35. Springer (2020)
35. GitHub: Github copilot - your ai pair programmer. `https://copilot.github.com/`. Accessed: 2022-05-31
36. Hsu, C.W., Chang, C.C., Lin, C.J., et al.: A practical guide to support vector classification (2003)
37. Hutchinson, J., Rouncefield, M., Whittle, J.: Model-driven engineering practices in industry. In: International Conference on Software Engineering, pp. 633–642. ACM, Waikiki, Honolulu, Hawaii (2011)
38. Kappel, G., Langer, P., Retschitzegger, W., Schwinger, W., Wimmer, M.: Model transformation by-example: a survey of the first wave. In: Conceptual Modelling and Its Theoretical Foundations, pp. 197–215. Springer, New York, NY, USA (2012)
39. Kotsiantis, S.B., Zaharakis, I., Pintelas, P.: Supervised machine learning: A review of classification techniques. Emerging artificial intelligence applications in computer engineering **160**(1), 3–24 (2007)
40. Kumar, M.A.: Efficient weight assignment method for detection of clones in state flow diagrams. Journal of Software Engineering Research and Practices **4**(2), 12–16 (2014)
41. Kuschke, T., Mäder, P.: Pattern-based auto-completion of uml modeling activities. In: Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering, ASE '14, p. 551–556. Association for Computing Machinery, New York, NY, USA (2014). DOI 10.1145/2642937.2642949. URL `https://doi.org/10.1145/2642937.2642949`
42. Kuschke, T., Mäder, P., Rempel, P.: Recommending auto-completions for software modeling activities. In: International Conference on Model Driven Engineering Languages and Systems, pp. 170–186. Springer (2013)
43. Liu, B., Hsu, W., Ma, Y., et al.: Integrating classification and association rule mining. In: Kdd, vol. 98, pp. 80–86 (1998)
44. Mazanek, S., Maier, S., Minas, M.: Auto-completion for diagram editors based on graph grammars. In: IEEE Symposium on Visual Languages and Human-Centric Computing, pp. 242–245. IEEE (2008)
45. Mussbacher, G., Combemale, B., Abrahão, S., Bencomo, N., Burgueño, L., Engels, G., Kienzle, J., Kühn, T., Mosser, S., Sahraoui, H., et al.: Towards an assessment grid for intelligent modeling assistance. In: Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings, pp. 1–10 (2020)
46. Mussbacher, G., Combemale, B., Kienzle, J., Abrahão, S., Ali, H., Bencomo, N., Búr, M., Burgueño, L., Engels, G., Jeanjean, P., et al.: Opportunities in intelligent modeling assistance. Software and Systems Modeling **19**(5), 1045–1053 (2020)
47. Nair, A., Ning, X., Hill, J.H.: Using recommender systems to improve proactive modeling. Software and Systems Modeling pp. 1–23 (2021)

48. Pati, T., Feiock, D.C., Hill, J.H.: Proactive modeling: auto-generating models from their semantics and constraints. In: Workshop on Domain-specific modeling, pp. 7–12. ACM (2012)
49. Petersen, H.: Clone detection in Matlab Simulink models. Master's thesis, Technical University of Denmark, 2012, iMM-M. Sc.-2012-02 (2012)
50. Pham, N., Nguyen, H., Nguyen, T., Al-Kofahi, J., Nguyen, T.: Complete and accurate clone detection in graph-based models. In: International Conference on Software Engineering (ICSE), pp. 276–286 (2009)
51. Proksch, S., Lerch, J., Mezini, M.: Intelligent code completion with bayesian networks. ACM Trans. Softw. Eng. Methodol. **25**(1), 1–31 (2015). DOI 10.1145/2744200
52. Raychev, V., Vechev, M., Yahav, E.: Code completion with statistical language models. In: Conference on Programming Language Design and Implementation, pp. 419–428. ACM, New York, NY, USA (2014). DOI 10.1145/2594291.2594321
53. Reicherdt, R., Glesner, S.: Slicing matlab simulink models. In: International Conference on Software Engineering, pp. 551–561. IEEE Press (2012)
54. Robbes, R., Lanza, M.: Improving code completion with program history. Automated Software Engineering **17**(2), 181–212 (2010)
55. Robillard, M.P., Maalej, W., Walker, R.J., Zimmermann, T.: Recommendation systems in software engineering. Springer Science & Business (2014)
56. Roy, C.K., Cordy, J.R.: A survey on software clone detection research. Tech. Rep. 2007-541, Queen's University (2007)
57. Roy, C.K., Cordy, J.R.: A mutation/injection-based automatic framework for evaluating code clone detection tools. In: International Conference on Software Testing, Verification and Validation Workshops (ICSTW), pp. 157–166 (2009)
58. Rubin, J., Chechik, M.: N-way model merging. In: proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, pp. 301–311 (2013)
59. Sagi, O., Rokach, L.: Ensemble learning: A survey. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery **8**(4), e1249 (2018)
60. Saini, R., Mussbacher, G., Guo, J.L.C., Kienzle, J.: Domobot: A bot for automated and interactive domain modelling. In: Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings, MODELS '20. Association for Computing Machinery, New York, NY, USA (2020). DOI 10.1145/3417990.3421385. URL `https://doi.org/10.1145/3417990.3421385`
61. Schäfer, M., Sridharan, M., Dolby, J., Tip, F.: Effective smart completion for javascript. Technical Report RC25359 (2013)
62. Segura, A.M., de Lara, J.: Extremo: An eclipse plugin for modelling and meta-modelling assistance. Science of Computer Programming **180**, 71 – 80 (2019). DOI https://doi.org/10.1016/j.scico.2019.05.003. URL `http://www.sciencedirect.com/science/article/pii/S0167642319300644`
63. Segura, Á.M., Pescador, A., de Lara, J., Wimmer, M.: An extensible meta-modelling assistant. In: International Enterprise Distributed Object Computing Conference, pp. 1–10 (2016)
64. Sen, S., Baudry, B., Vangheluwe, H.: Towards domain-specific model editors with automatic model completion. Simulation **86**(2), 109–126 (2010)
65. Steimann, F., Ulke, B.: Generic model assist. In: International Conference on Model Driven Engineering Languages and Systems, pp. 18–34. Springer (2013)
66. Stephan, M.: Model clone detector evaluation using mutation analysis. In: 2014 IEEE International Conference on Software Maintenance and Evolution, pp. 633–638. IEEE (2014)
67. Stephan, M.: Towards a Cognizant Virtual Software Modeling Assistant Using Model Clones. In: Proceedings of the 41st International Conference on Software Engineering: New Ideas and Emerging Results, ICSE-NIER '19, pp. 21–24. IEEE Press, Piscataway, NJ, USA (2019). DOI 10.1109/ICSE-NIER.2019.00014
68. Stephan, M., Alalfi, M., Cordy, J.R.: Towards a taxonomy for simulink model mutations. In: International Workshop on Mutation Analysis, pp. 206–215 (2014)
69. Stephan, M., Cordy, J.R.: A survey of model comparison approaches and applications. In: International Conference on Model-Driven Engineering and Software Development, pp. 265–277 (2013)
70. Stephan, M., Cordy, J.R.: Mumonde: A framework for evaluating model clone detectors using model mutation analysis. Software Testing, Verification and Reliability p. e1669 (2018)
71. Storrle, H.: Towards clone detection in UML domain models. Software & Systems Modeling **12**(2), 307–329 (2013)
72. Voorhees, E.M., et al.: The trec-8 question answering track report. In: Trec, vol. 99, pp. 77–82 (1999)
73. Weyssow, M., Sahraoui, H., Syriani, E.: Recommending metamodel concepts during modeling activities with pre-trained language models. Software and Systems Modeling **21**(3), 1071–1089 (2022)
74. Zhang, C., Ma, Y.: Ensemble machine learning: methods and applications. Springer (2012)