# Traceability Link Recovery between Requirements and Models using an Evolutionary Algorithm Guided by a Learning to Rank Algorithm: Train Control and Management Case

Ana C. Marcén[a,b], Raúl Lapeña[a], Óscar Pastor[b], Carlos Cetina[a]

*{acmarcen,rlapena,ccetina}@usj.es, opastor@pros.upv.es*

[a]*SVIT Research Group*
*Universidad San Jorge*
[b]*Centro de Investigación en Métodos de Producción de Software*
*Universitat Politècnica de València*

## Abstract

Traceability Link Recovery (TLR) has been a topic of interest for many years within the software engineering community. In recent years, TLR has been attracting more attention, becoming the subject of both fundamental and applied research. However, there still exists a large gap between the actual needs of industry on one hand and the solutions published through academic research on the other.

In this work, we propose a novel approach, named Evolutionary Learning to Rank for Traceability Link Recovery (TLR-ELtoR). TLR-ELtoR recovers traceability links between a requirement and a model through the combination of evolutionary computation and machine learning techniques, generating as a result a ranking of model fragments that can realize the requirement.

TLR-ELtoR was evaluated in a real-world case study in the railway domain, comparing its outcomes with five TLR approaches (Information Retrieval, Linguistic Rule-based, Feedforward Neural Network, Recurrent Neural Network, and Learning to Rank). The results show that TLR-ELtoR achieved the best results for most performance indicators, providing a mean precision value of 59.91%, a recall value of 78.95%, a combined F-measure of 62.50%, and a MCC value of 0.64. The statistical analysis of the results assesses the magnitude of the improvement, and the discussion presents why TLR-ELtoR achieves better results than the baselines.

*Keywords:* Traceability Link Recovery, Requirements Engineering, Models, Evolutionary Algorithm, Learning to Rank

## 1. Introduction

Traceability Link Recovery (TLR) has been a subject of investigation for many years within the software engineering community [1, 2]. Research has shown that affordable traceability can be critical to the success of a project [3] and leads to increased maintainability and reliability of software systems by making it possible to verify and to trace non-reliable parts [4]. Specifically, more complete traceability decreases the expected defect rate in the developed software [5].

In recent years, TLR has been attracting more attention, and re-establishing the traceability links between software artifacts has become a subject of both fundamental and applied research [6]. In fact, a few approaches have been proposed to recover traceability between requirements and models [7]. However, the support of traceability research for practical problems in industry is perceived as being rather low [7], and there still exists a large gap between the needs of industry on one hand and the published solutions from academic research on the other [8].

In this work, we propose a novel approach, named Evolutionary Learning to Rank for Traceability Link Recovery (TLR-ELtoR). TLR-ELtoR recovers traceability links between the requirements of a software system and the models that implement it. Specifically, our approach is based on an Evolutionary Algorithm (EA). Moreover, the EA is guided by a Learning to Rank algorithm that empowers us to take advantage of the knowledge and the experience that have been generated in companies for years in order to automatically perform ranking tasks. In summary, from a requirement and a model, TLR-ELtoR generates a ranking of model fragments that can realize the requirement.

The presented approach was evaluated in a real-world case study provided by our industrial partner, CAF[1] (Construcciones y Auxiliar de Ferrocarriles), a worldwide provider of railway solutions. The outcomes of TLR-ELtoR were compared with five TLR approaches, these approaches were selected taking into account the approaches that obtain the best results for recovering trace-

[1]`www.caf.net/en`

ability between requirements and models [7], the successful application of deep learning techniques for TLR in recent works [9], and the size of the search space to be explored. The first one [10] is a Linguistic Rule-Based (TLR-Linguistic) approach that is based on Parts-of-Speech (POS) Tagging and traceability rules. The second one [11, 12] is an Information Retrieval (TLR-IR) approach that is based on Latent Semantic Indexing (LSI) and Singular Value Decomposition (SVD). The third one is a Feedforward Neural Network (TLR-FNN) approach that is based on a traditional neural network structure. The fourth one is a Recurrent Neural Network (TLR-RNN) approach that is based on an extension of a Feedforward Neural Network with feedbaack connections to model the temporal characteristics of the problem being learned [13]. The fifth one is a Learning to Rank (TLR-LtoR) approach based on ranking Machine Learning algorithms of the same name.

The results show that TLR-ELtoR achieved the best results for most the performance indicators, providing a mean precision value of 59.91%, a recall value of 78.95%, a combined F-measure of 62.50%, and a MCC value of 0.64. In contrast, the TLR-Linguistic baseline, the TLR-IR baseline, and the TLR-LtoR baseline had worse results for these same measurements. On the other hand, although TLR-FNN and TLR-RNN achieved the best results for recall, they obtained the worst results for the rest of the indicators. The statistical analysis of the results assesses the magnitude of the improvement, and the discussion presents two advantages and a limitation of our approach.

The remainder of this paper is structured as follows: Section 2 provides background on our case study. Section 3 highlights our TLR-ELtoR approach. Section 4 and Section 5 detail the genetic operations step and the fitness function step of our approach, respectively. Section 6 details the means used to evaluate our work and the results of the evaluation. Section 7 analyzes the statistical significance of the obtained results. Section 8 discusses our approach and the obtained results. Section 9 describes the threats to the validity of our work. Section 10 introduces the existing works that are related to our work. Section 11 provides the means to replicate the results. Finally, Section 12 concludes the paper.

## 2. Background

This section presents the Train Control and Management Language (TCML), which is used to formalize the products manufactured by our industrial partner. TCML has the expressiveness required to describe the interac-

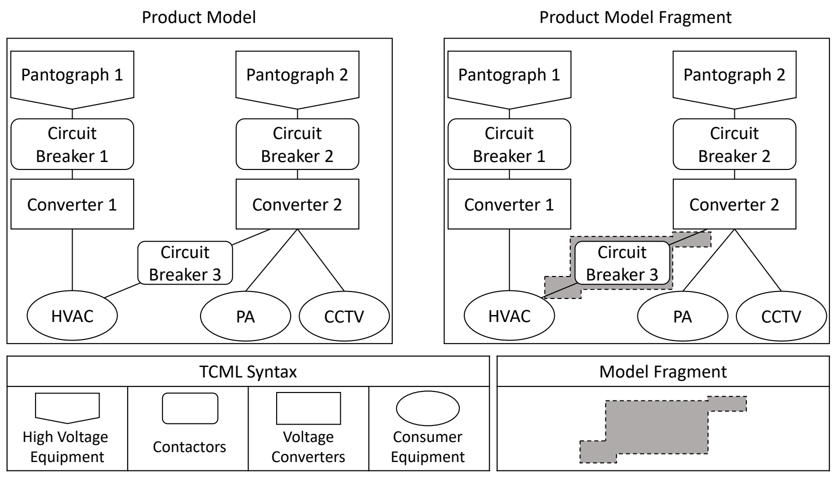


Figure 1: Example of a TCML model and model fragment

tion between the main pieces of equipment installed in a train unit. TCML also has the required expressiveness to specify non-functional aspects that are related to regulation, such as the quality of signals from the equipment or the different levels of installed redundancy. TCML will be used through the rest of the paper to present a running example. In this work, for the sake of the understandability and legibility of the running example, we present an equipment-focused, simplified subset of TCML, with four different kinds of equipment:

1 **High Voltage Equipment**, which is in charge of harvesting the energy that powers the different elements of the train.

2 **Contactors**, which are in charge of opening or closing the circuits between the High Voltage equipment and the Voltage Converters.

3 **Voltage Converters**, which are in charge of transforming the harvested electric power into a current that the Consumer Equipment can work with.

4 **Consumer Equipment**, which is in charge of carrying out all of the tasks required for the train to work properly and provide comfort to the passengers.

Figure 1 depicts an example that is taken from a real-world train. It presents a converter assistance scenario. In the example, two separate pantographs (High Voltage Equipment) collect energy from the overhead wires and send it to their respective circuit breakers (Contactors), which in turn send it to their independent Voltage Converters. The converters then power their assigned Consumer Equipment: the left one powers the HVAC (the air conditioning system of the train) devices, and the right one powers the PA (public address system) and the CCTV (television system) circuits of the train.

There is an additional circuit breaker between the second converter and the HVAC that is connected to the first converter. The part on the right of Figure 1 shows an example of a model fragment of the product model. The model fragment includes the additional circuit breaker. This model fragment is the realization of the "converter assistance" requirement, which allows the passing of current from one converter to a piece of Consumer Equipment that is assigned to its peer. In the case of overload or failure of the first converter, total or partial functioning (depending on specific conditions) of the HVAC could be covered by the second converter.

To formalize the model fragments, we use the Common Variability Language (CVL) [14]. CVL defines variants of a base model (conforming to MOF, the OMG metalanguage for defining modeling languages) by replacing variable parts of the base model with alternative model replacements that are found in a library.

## 3. Overview of our TLR-ELtoR Approach

This section presents the proposed TLR-ELtoR approach for TLR between the requirements and the models through an EA, which is based on genetic operations and a fitness function. The objective of the approach is to provide the model fragment from a given model that realizes a specific requirement. To do this, the approach receives as input the model that implements a specific requirement. The approach relies on an evolutionary algorithm that iterates over a population of model fragments, evolving them using genetic operations. Then, the score of each model fragment and its position in the ranking are calculated through the fitness function that uses Learning to Rank as its objective. As output, the approach provides a model fragment ranking where each model fragment is ranked taking into account how well the model fragment implements the input requirement.

The top of Figure 2 shows an example of input to our approach: the model that contains the requirement and the requirement description, which uses natural language to define the target requirement. The center of Figure 2 shows a simplified representation of the main steps of our approach. Rounded rectangles represent the different steps of the approach, and straight rectangles represent the inputs and outputs of each of the steps. Our approach has three steps:

1 **Initialization:** The first step is to generate a population of model fragments from the model, which serves as input for the evolutionary algorithm. In order to generate the population of model fragments, parts of the model are extracted randomly and added to a collection of model fragments.

2 **Genetic operations:** Second, genetic operations are applied to the model fragment population in order to generate candidate model fragments for the target requirement.

3 **Fitness function:** Finally, the new model fragment population is assessed through the fitness function, which evaluates each of the generated model fragments.

The last two steps of the approach are repeated until the solution converges to a certain stop condition. Usually, the stop condition can be a time slot, a fixed number of generations, or a trigger value of the fitness that makes the process finish when reached [15]. Since, the stop condition greatly depends on the domain and the problem being solved, it is adjusted depending on the results being output, taking into account when the fitness values are converging and no further improvements are being made by new generations [15]. When the stop condition is met, the evolutionary algorithm provides a model fragment list, which is ranked according to the objectives for the requirement (see the bottom of Figure 2).

The following sections describe the genetic operations of TLR-ELtoR for generating new model fragments and how the fitness of each model fragment is determined in terms of similarity to the requirement description.

## 4. Genetic Operations of the TLR-ELtoR Approach

The second step of our approach is to generate and to evolve a set of model fragments that could realize the requirement. To do this, this step

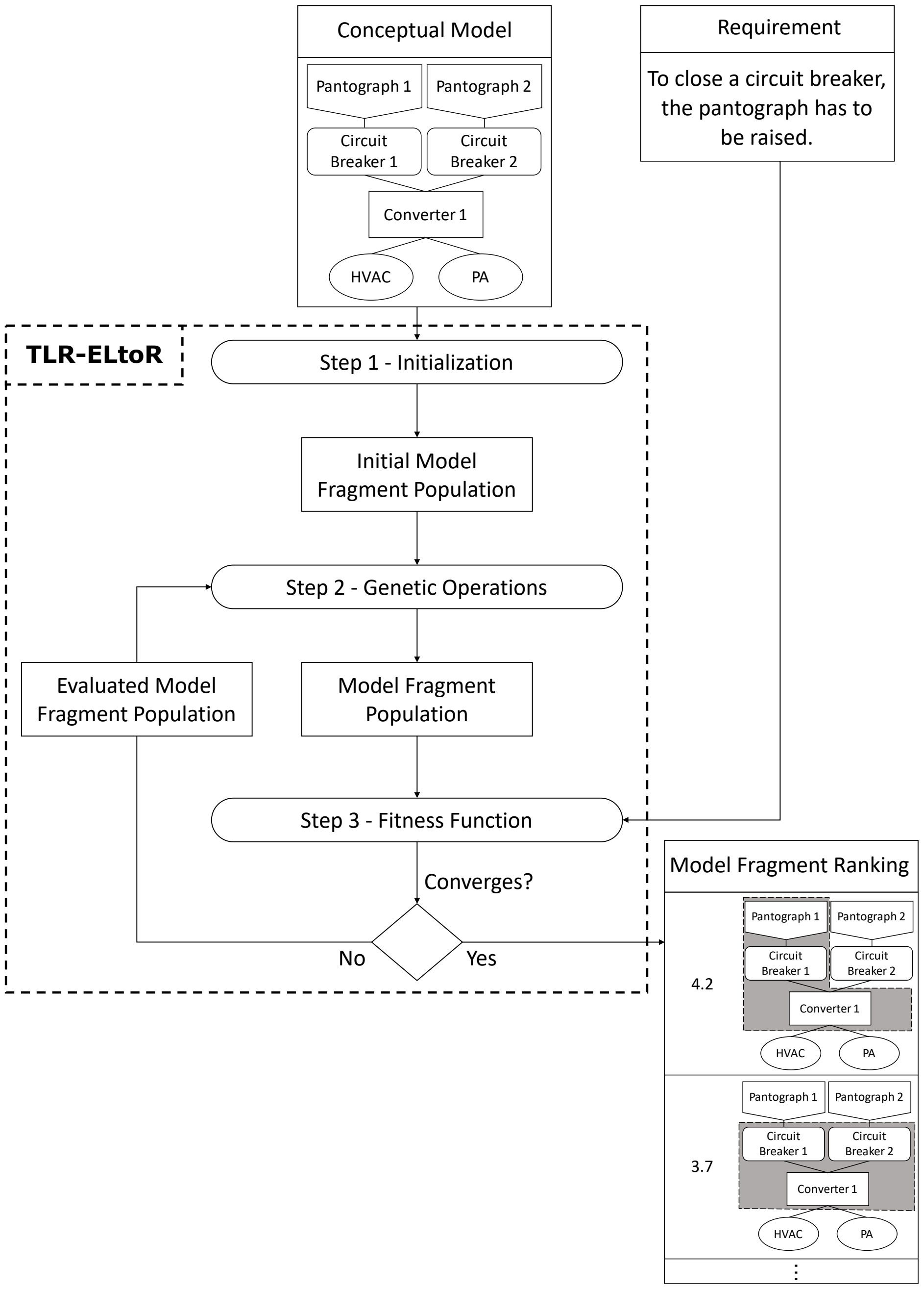
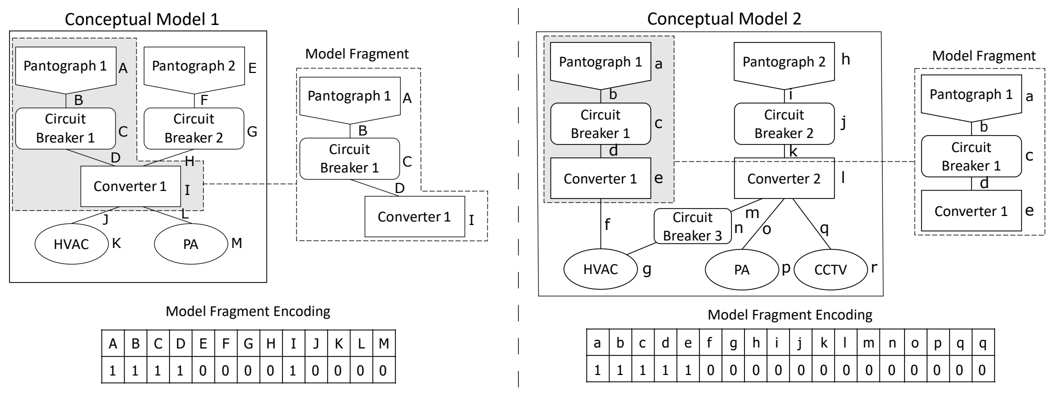

Conceptual Model
Pantograph 1
Pantograph 2
Circuit Breaker 1
Circuit Breaker 2
Converter 1
HVAC
PA
Requirement
To close a circuit breaker, the pantograph has to be raised.
TLR-ELtoR
Step 1 - Initialization
Initial Model Fragment Population
Step 2 - Genetic Operations
Evaluated Model Fragment Population
Model Fragment Population
Step 3 - Fitness Function
Converges?
No
Yes
Model Fragment Ranking
4.2
3.7

Figure 2: Overview of the approach



| A | B | C | D | E | F | G | H | I | J | K | L | M |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

| a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | q |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Figure 3: Examples of Model Fragment Encoding

deals with the encoding of the model fragments and the selection of genetic operators.

## *4.1. Model Fragment Encoding for Genetic Operations*

Traditionally, in evolutionary algorithms, each possible solution of the problem is encoded as a string of binary values. However, encoding each model fragment as a string of binary values is not straightforward. The authors in [16] propose an encoding where each model fragment is encoded as an individual in relation to the model. In other words, each individual is a set of model elements that are present or absent in a model fragment.

Figure 3 shows two examples of the representation of model fragments. Each letter labels a model element of the model. Therefore, the individual contains as many positions as model elements in the model and the binary value of these positions depends on the presence or absence of the model elements in the model fragment. If the model element appears in the model fragment, the value will be 1; if the model element does not appear in the model fragment the value will be 0.

Figure 3 also shows that the encoding will be different for different models, even though the model fragment to be encoded is the same. Both of the examples in Figure 3 represent the same model fragment. However, since they come from different models, their representations are different.

## *4.2. Genetic Operators*

The generation of new model fragments (based on existing ones) is done by applying a set of three genetic operators, which are adapted to work on model fragments. These genetic operations were introduced for the first time in [15] to carry out the selection of parents, the crossover, and the mutation of model fragments.

The **selection operator** picks the best candidates from the population as input for the rest of the operators. There are different methods that can be used to perform the selection of the parents. One of the most widespread methods (adopted by our work) is to follow the wheel selection mechanism [17], where each model fragment from the population has a probability of being selected that is proportional to its fitness score. Candidates with high fitness values have higher probabilities of being chosen as parents for the next generation.

The **crossover operation** enables the creation of two new individuals by combining the genetic material from two model fragments. A randomly generated mask determines how the combination is done, indicating for each element of the model fragments if the offspring should inherit from one model fragment or the other. Specifically, the mask is created randomly and all of the model elements have a 50% probability of belonging to the mask. To do this, a random number (0 or 1) is generated for each model element. The elements whose value is 1 belong to the mask and the elements whose value is 0 belong to the inverse of the mask. Moreover, a model fragment is a subset of the elements that are present in a model. Since both model fragments are extracted from the same model, their combination will always return a model fragment that is part of the original product model. As a result of the **crossover operation**, two individuals are generated: one by directly applying the mask, and the other one by applying the inverse of the mask, as is usually done in genetic algorithms [18].

The **mutation operator** is used to imitate the mutations that occur randomly in nature when new individuals are born. In other words, new individuals have small differences with their parents that could make them adapt better (or worse) to their living environment. Following this idea, the mutation operator applied to model fragments [15] takes as input a model fragment and mutates it into a new one, which is returned as output. Specifically, the mutation operator can perform two kinds of modifications: the addition of elements to the model fragment, or the removal of elements from the model fragment. Since the approach is looking for fragments of the model that realize a specific requirement, the new modified fragment must remain a part of this model. Therefore, the modifications that can be done to the model fragment must be driven by the model, which determines the additions

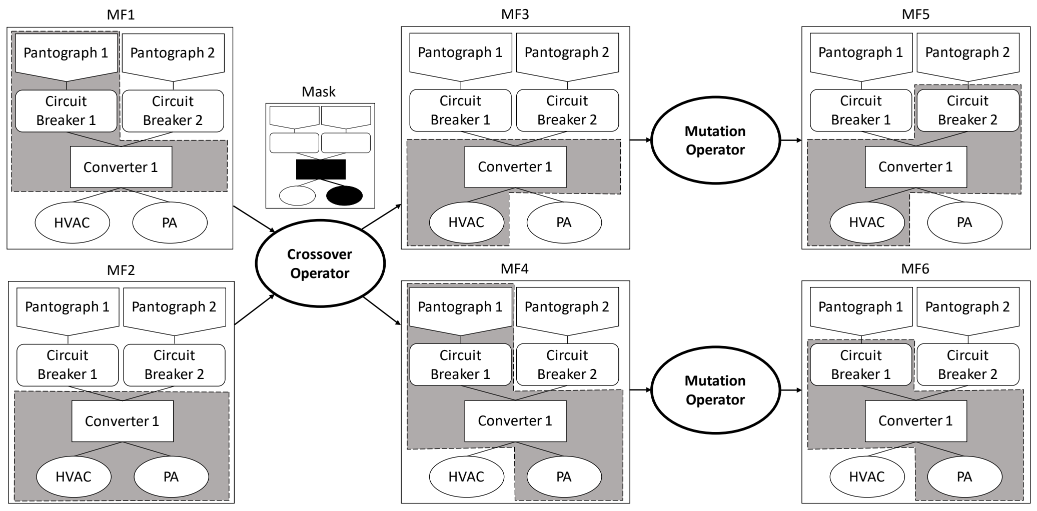


Figure 4: Example of genetic operations

and subtractions of elements that can be applied to the model fragments in the population.

After applying the genetic operators, it may be that not all of the elements of the new individuals are connected. Indeed, the requirement can be implemented by several model elements that are not directly connected in the model [15]. Therefore, it is necessary to create fragments of this kind since they could be the ones realizing the requirement.

Figure 4 shows an example of the application of the two genetic operations. First, the crossover operation is applied. We select the two model fragments to which the operator is applied. Then, the first model fragment (MF1) is combined with the second model fragment (MF2) according to a mask that contains two sets of elements (one regular and one marked in black). To create the first of the new individuals, we interpret the mask by selecting the blackened elements from the first parent (MF1) and the regular elements from the second parent (MF2). As a result, the new model fragment (MF3) contains the set of elements that are present in the mask in MF1 and the set of elements that are absent in the mask in MF2. In addition, the mask is also interpreted in the opposite way by selecting the blackened elements from MF2 and the regular elements from MF1, thus producing another new and distinct model fragment (MF4).

Afterwards, the mutation operator is applied. In this example, the mutation operation takes the first offspring produced through the crossover operator and adds one element (the second circuit breaker). Then, the mutation operation takes the second offspring and removes one element (the first pantograph). The resulting model fragments (MF5 and MF6) are new candidates in the population for the realization of the input requirement.

## 5. Fitness of the TLR-ELtoR Approach

In evolutionary algorithms, this step determines what degree of adaptation to the environment each individual has. Following this idea, in our approach, the fitness step is used to assess how suitable each model fragment is in comparison with the target requirement. To do this, a Learning to Rank algorithm is used to rank a set of model fragments depending on their closeness to a requirement.

Learning to Rank (LtoR) is the name given to a family of Machine Learning algorithms, which automatically address ranking tasks. Specifically, the LtoR algorithms make possible the construction of a classifier that contains a set of rules to rank objects. The classifier automatically learns these rules by comparing the objects within a knowledge base. Then, since the classifier knows how to rank objects following these learned rules, the classifier can be used to rank new objects. In other words, Learning to Rank algorithms use a knowledge base to generate a classifier, which is called training. Then, the classifier is used to rank new objects, which is called testing [19].

Figure 5 shows the overview of the Fitness Function, where the LtoR algorithm is applied in our approach. The part on the left of the figure shows the training process where the classifier learns how well each model fragment realizes a specific requirement. To do this, the knowledge base contains traces between requirements and model fragments that are known. The part on the right of the figure shows the testing process where a population of model fragments is ranked by means of the classifier, which determines which model fragment is a better realization of the requirement than another model fragment. Therefore, the classifier is considered as both an artifact (output from the training process) and a step (responsible for ranking in the testing process). For this reason, Figure 5 shows the classifier in a black, rounded rectangle to point out its double meaning. The following sections provide a more detailed description of the training and testing processes. However, since both training and testing have to encode their inputs (see Figure 5), the encoding is explained before these processes.

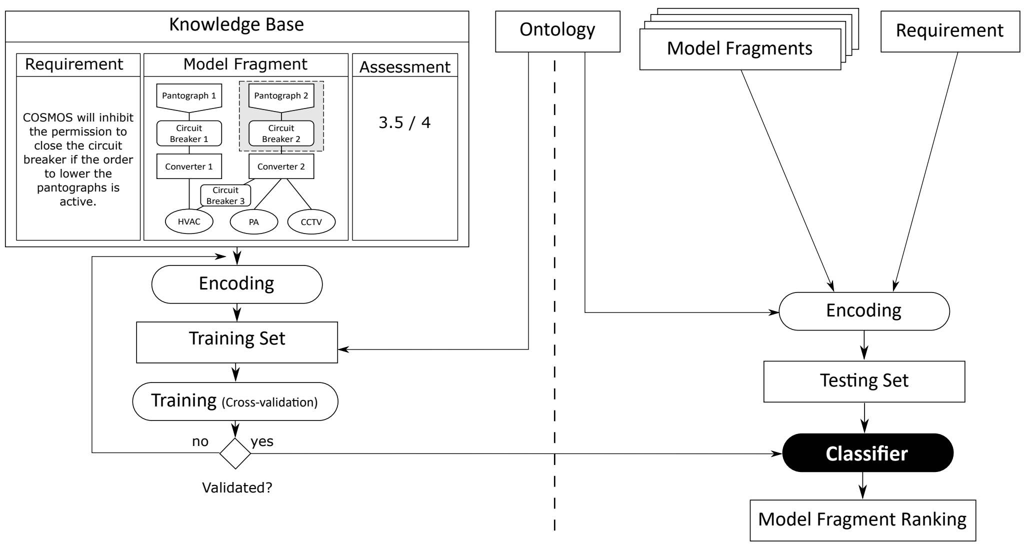


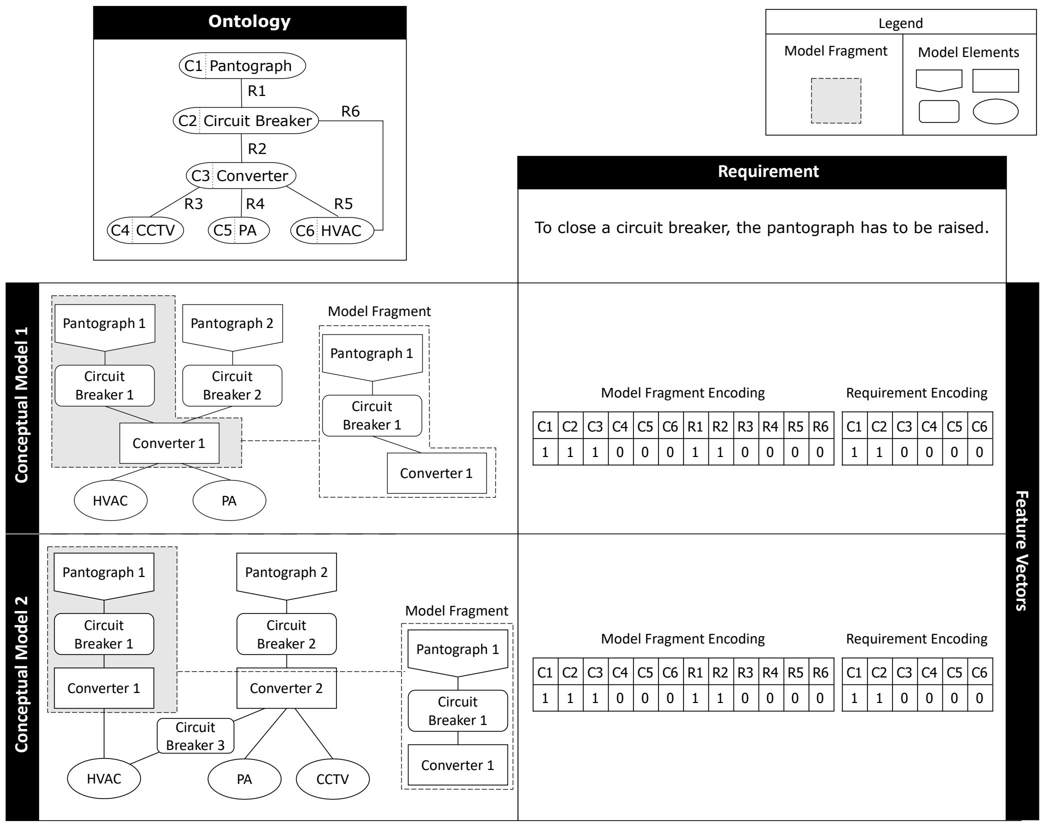
Figure 5: Overview of the Fitness Function

## *5.1. Model Fragment Encoding for the Fitness Function*

Since most of the Machine Learning techniques, such as Learning to Rank algorithms, are designed to process feature vectors as inputs [20], our model fragments have to be encoded into feature vectors to be able to use LtoR. Feature vectors are known as the ordered enumeration of features that characterize the object being observed [21].

However, the set of features that are selected to characterize the object have to be the same for all of the model fragments. This guarantees that the feature vectors have the same length and the same features so that the comparison between feature vectors is fair. Moreover, the fitness function determines the suitability between the model fragment and the target requirement. Since both the model fragment and the requirement are being observed, the feature vectors would have to be generated by encoding them both. For these reasons, the encoding for the fitness function is different from the encoding for genetic operations where both the length and the values of the individuals depend on model elements and the requirement is not considered.

In [22], we proposed an encoding where each model fragment is encoded as a feature vector taking into account an ontology. Specifically, each concept and relation in the ontology is represented as a feature in the feature vector. The value of each feature is computed as the frequency of the concept or



Model Fragment Encoding (Conceptual Model 1)

| C1 | C2 | C3 | C4 | C5 | C6 | R1 | R2 | R3 | R4 | R5 | R6 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |

Requirement Encoding (Conceptual Model 1)

| C1 | C2 | C3 | C4 | C5 | C6 |
|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 | 0 |

Model Fragment Encoding (Conceptual Model 2)

| C1 | C2 | C3 | C4 | C5 | C6 | R1 | R2 | R3 | R4 | R5 | R6 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |

Requirement Encoding (Conceptual Model 2)

| C1 | C2 | C3 | C4 | C5 | C6 |
|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 | 0 |

Figure 6: Examples of Encoding for Model Fragments and Requirements in the Fitness step

the relation in the model fragment. In the same way, the target requirement is encoded as part of the feature vector taking into account the ontology. Specifically, each concept in the ontology is represented as a feature in the feature vector and the value of each feature is computed as the frequency of the concept in the requirement.

Since both requirements and model fragments are based on natural language, the terms used in the ontology do not always align well with the terms in the requirements and with the terms in the model fragments. For this reason, Natural Language Processing (NLP) techniques are used to process both the requirements and the model fragments before applying the encoding. Specifically, the requirements and the model fragments are processed by a combination of NLP techniques defined in [23], which consists of tokenizing, lowercasing, removal of duplicate keywords, syntactical analysis, lemmatization, and stopword removal.

Figure 6 shows two examples of the representation of two model fragments and the target requirement. The concepts and relations of the ontology are features in the feature vector. For example, the concept *Pantograph* is mapped as *C1*, and the relation between the concepts *Converter* and *HVAC* is mapped as *R5*. On the one hand, these concepts and relations are compared with the model fragments, so their values correspond to the number of occurrences of each concept or relation in the model fragment. Therefore, for the first model, the value of the feature *C1* is 1 because there is one pantograph in the model fragment, and the value of the feature *R5* is 0 because there is no relation of the type *Converter-HVAC* in the model fragment. In fact, both models contain a relation of the type *Converter-HVAC*, but none of the model fragments contain this relation. Therefore, the value for *R5* is 0 in both feature vectors because each feature vector only contains the encoding of a model fragment, not the encoding of the whole model. On the other hand, the concepts are also compared with the target requirement, so their values correspond to the number of occurrences of each concept in the requirement. Therefore, the value of the *C1* is 1 because the concept pantograph appears once in the requirement. Figure 6 also shows that the feature vectors do not depend on whole models because they only represent the encoding of model fragments and requirements. Therefore, two model fragments may be result in the same feature vector although their models are different.

## *5.2. Training Process*

The target of the training process is to produce a classifier from a training set, which ranks the model fragments generated by the genetic operations. To do this, the knowledge base has to be encoded to obtain the training set, which is used in a LtoR algorithm to generate a classifier. Also, before using this classifier in the testing process, the validation of its performance is a good practice in order to improve the results.

The knowledge base is composed of traces between requirements and model fragments that are known. Specifically, the knowledge base consists of a set of traces that are generated using the domain experts' experience, results, and documentation, where each trace of the knowledge base is composed of a requirement, a model fragment, and an assessment. The requirement uses natural language to define the requirement. The model fragment consists of an element or a set of elements that belongs to a model. The assessment determines if the model fragment realizes the requirement to a greater or lesser extent. In other words, the assessment determines the similarity between the requirement and the model fragment. Figure 5 shows an example of the knowledge base for performing requirement traceability.

In order to apply LtoR algorithms in models, the first step consists of encoding the traces of the knowledge base into the feature vectors. Therefore, the model fragment and the requirement of each trace are encoded following the encoding for the fitness function (see Section 5.1), and the assessment is also included as part of the encoding. Then, the obtained feature vectors compose the training set.

The training set is used to train a classifier, which learns a rule-set through the comparison of the feature vectors of the training set [24]. However, before using this classifier to rank the model fragment in the testing process, it is worth analyzing the performance of the classifier through cross-validation. Cross-validation is a statistical method of evaluating and comparing ML algorithms by dividing data into two segments: one used to train a classifier, and the other used to validate the classifier [25]. Moreover, to reduce variability, multiple rounds of cross-validation are performed using different partitions, and the results are averaged over the rounds [26].

The results of the cross-validation provide the performance of the classifier. If this performance is not considered suitable, it is necessary to perform another training iteration. In this iteration, some artifacts of the training process (e.g., the encoding, the ontology, the knowledge base, or the LtoR algorithm) have to be modified in order to improve the classifier. Otherwise, if the performance is considered suitable, the classifier obtains the go-ahead, so the classifier trained with the whole knowledge base is used in the testing process. Once the classifier has been generated, the training process does not have to be repeated again. The same classifier is used whenever the Fitness Function is applied. Therefore, the training process is only performed in the first iteration of the EA, when there is not yet a classifier.

## *5.3. Testing Process*

In our approach, the classifier is used to rank the model fragments that are generated after each iteration of the EA. Specifically, the classifier assigns a score to each model fragment based on its closeness to the requirement. For example, taking into account the requirement of Figure 6, the model fragments must contain at least one *circuit breaker* so that these model fragments are close to this requirement. Therefore, the model fragment with a *circuit breaker* would obtain a better fitness score than a model fragment without a *circuit breaker*.

However, before ranking the model fragments, both the model fragments and the requirement have to be encoded into feature vectors (see Section 5.1) so that the classifier can understand them. The feature vectors obtained from the encoding compose the testing set. Then, each feature vector of the testing set is tested by the classifier, which used the learned rule-set in the training process to assign a fitness score to each one of them.

The fitness score is a numerical value that is greater than 0. If the fitness score is close to 0, the model fragment is not close to the requirement, so the model fragment is not relevant to the requirement. In contrast, the greater the fitness score, the more relevant the model fragment is. Taking into account the fitness scores, the model fragments can be ordered in a ranking where the top positions are occupied by the model fragments with the highest relevance to the requirement.

Finally, as a result, this ranking of model fragments is returned by the Fitness Function, (see Figure 2). Therefore, in each iteration of the EA, the Fitness Function provides a ranking of model fragments organized by their fitness score. Then, if the stop condition is satisfied, this ranking of model fragment is obtained as a result of the approach. However, if the stop condition is not satisfied, a new iteration is performed. Therefore, the model fragments are considered as the new population to be evolved; then the genetic operations select the best model fragments, mutate them, and create new model fragments from them.

## 6. Evaluation

This section presents the evaluation of our approach: the experimental setup, the baselines, a description of the case study where we applied the evaluation, the implementation details, and the obtained results.

### *6.1. Experimental Setup*

The goal of this experiment is to perform TLR between requirements and models through TLR-ELtoR and to compare the results with the TLR approaches that have obtained the best results in the literature. Figure 7 shows an overview of the process that was followed to evaluate our approach (TLR-ELtoR) and the baselines (TLR-Linguistic, TLR-IR, TLR-FNN, TLR-RNN, TLR-LtoR). The top part of Figure 7 shows the inputs, which are extracted from the documentation provided by our industrial partner: knowledge base, ontology, requirements, product models, and approved traceability between requirements and product models. Each test case is comprised of a requirement, a model of a product, the ontology, and the knowledge base. However, the ontology and the knowledge base are ignored by TLR-Linguistic and TLR-IR because they do not need it. The Oracle is composed of the approved traceability between the requirements and the models.

For each test case, our approach generates a ranking of model fragments. Each model fragment contains the elements of the model that are related to the requirement, so each model fragment fits the traceability between the model and the requirement to a greater or lesser extent. Then, we take the best solution of the ranking and compare it against the oracle, which is the ground truth. Once the comparison is performed, a confusion matrix is calculated. The baselines also recover the traceability links between the requirement and the model in the test cases. As a result, each baseline generates a model fragment. These model fragments are also compared with the oracle, and a confusion matrix is calculated for each baseline. Therefore, we obtain six confusion matrices, one for our TLR-ELtoR approach and one for each baseline.

A confusion matrix is a table that is often used to describe the performance of a classification model (in this case, both the TLR-ELtoR and the baselines) on a set of test data (the solutions) for which the true values are known (from the oracle). In our case, each solution that is outputted by the approaches is a model fragment that is composed of a subset of the model elements that are part of the product model. Since the granularity is at the level of model elements, the presence or absence of each model element is considered as a classification. The confusion matrix distinguishes between the predicted values and the real values, classifying them into four categories:

- True Positive (TP): values that are predicted as true (in the solution) and are true in the real scenario (the oracle).
- False Positive (FP): values that are predicted as true (in the solution) but are false in the real scenario (the oracle).

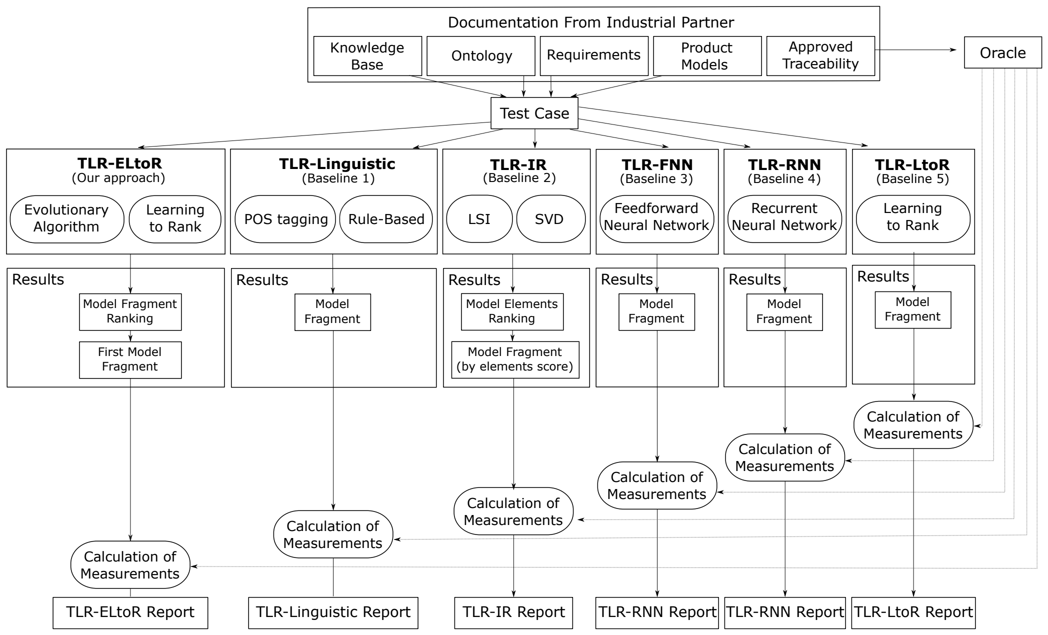


Figure 7: Experimental Setup

- True Negative (TN): values that are predicted as false (in the solution) and are false in the real scenario (the oracle).
- False Negative (FN): values that are predicted as false (in the solution) but are true in the real scenario (the oracle).

Then, some performance measurements are derived from the values in the confusion matrix. Specifically, we create a report that includes four performance measurements (recall, precision, the F-measure, and the Matthews Correlation Coefficient) for the test case for both the TLR-ELtoR and the baselines.

Recall measures the proportion of elements of the solution that are correctly retrieved by the proposed solution and is defined as follows:

$$Recall = \frac{TP}{TP + FN}$$

Precision measures the proportion of elements from the solution that are correct according to the ground truth (the oracle) and is defined as follows:

$$Precision = \frac{TP}{TP + FP}$$

The F-measure corresponds to the harmonic mean of precision and recall and is defined as follows:

$$F - measure = 2 * \frac{Precision * Recall}{Precision + Recall} = \frac{2 * TP}{2TP + FP + FN}$$

However, none of these previous measures correctly handle negative examples (TN). The **MCC** is a correlation coefficient between the observed and predicted binary classifications that takes into account all of the observed values (TP, TN, FP, FN) and is defined as follows:

$$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

Recall values can range between 0% (i.e., no single model element from the realization of the requirement obtained from the oracle is present in the model fragment of the solution) and 100% (i.e., all of the model elements from the oracle are present in the solution). Precision values can range between 0% (i.e., no single model element from the solution is the oracle) and 100% (i.e., all of the model elements from the solution are present in the oracle). A value of 100% precision and 100% recall implies that both the solution and the requirement realization from the oracle are the same. MCC values can range between $-1$ (i.e., there is no correlation between the prediction and the solution) to 1 (i.e., the prediction is perfect). Moreover, a MCC value of 0 corresponds to a random prediction.

#### *6.1.1. TLR-ELtoR Setup*

This section describes the technical details of our TLR-ELtoR approach taking into account the experimental setup defined. Specifically, four technical details are addressed: the stop condition, the hyperparameters for the evolutionary algorithm, the LtoR algorithm with its setting parameters, and the cross-validation method.

In general, there are two atomic performance measures for evolutionary algorithms: one regarding solution quality, and one regarding algorithm speed or search effort. In this paper, we focus on the solution quality (i.e., obtaining a solution that is more similar to the one from the oracle in terms of precision and recall). After running some prior tests to determine the number of iterations to converge (and adding a margin to ensure convergence), we allocated a fixed amount of iterations (200 iterations) to stop the execution.

For the settings of the evolutionary algorithm, namely population size, crossover probability, and mutation probability, we have chosen the values 100, 0.9, and 0.1, respectively. These were selected based on the parameters that are commonly used in the literature [27] and the results of some preliminary tuning experiments.

With regard to the LtoR algorithm, the selection of this algorithm depends on several aspects, such as the size of the knowledge base. RankBoost [28] belongs to the family of LtoR and is well known for its efficiency and effectiveness in different domains [29, 30]. Moreover, Rankboost can benefit from a small knowledge base together with a small number of features in the encoding to reduce the overfitting problem [31, 32]. Since this condition is satisfied by our case study, TLR-ELtoR was guided by Rankboost with the parameters tuned as in [33]. First, a grid search was built to determine the values of the parameters: number of iterations, and metric. Then, we uniformly sampled each of the two parameters in their range and evaluated all of the combinations of the sampled values. As a result, the parameters were tuned with *iteration* = 200 and *metric* equal to ERR10.

Moreover, even though our approach considers cross-validation as a step of the Fitness Function, the approach does not restrict the possibilities to one specific cross-validation method. In this evaluation, TLR-ELtoR used a $k$-fold validation with a $k$ value equal to 4. The $k$-fold validation is the most popular cross-validation procedure. Specifically, this method consists of randomly dividing the knowledge base into $k$-independent partitions. Then, $k - 1$ of the partitions are used to train the classifier, and this classifier is then used to test the partition that is left out. This procedure is repeated $k$ times, each time leaving out another partition. This produces $k$ estimations of the classifier, allowing assessment of its central tendency and variance [34].

## *6.2. Baselines*

Winkler et al. [7] classify several approaches that have been created over the past 15 years that try to optimize the automatic identification of traces. Based on this classification, as baselines, we selected the two approaches that obtain the best results for traceability links between requirements and models: (1) a rule-based approach that deduces traces by applying rules

(TLR-Linguistic) [10]; and (2) an information retrieval approach that can detect candidate traceability links through Information Retrieval (TLR-IR) [11, 12].

Deep learning techniques have also successfully been applied in TLR in some recent works [9]. Therefore, we decided to compare our approach with two baselines that apply deep learning: (1) the first one is based on a Feed-forward Neural Network (TLR-FNN); and (2) the second one is based on a Recurrent Neural Network (TLR-RNN).

Finally, to check the need for the evolutionary algorithm in our approach, TLR-ELtoR is also compared to TLR-LtoR, which explores the search space by means of brute-force. Therefore, the model fragments are generated from the model and evaluated through LtoR, but the results obtained from the LtoR process are not used to guide the generation of new model fragments. Since there is no guide to explore the model, the search for the model fragment that realizes a specific requirement is performed by brute-force.

#### *6.2.1. TLR-Linguistic: Linguistic Rule-Based Baseline*

Spanoudakis et al. [10] present a linguistic rule-based approach to support the automatic generation of traceability links between requirements and models. Specifically, the traceability links are generated following two stages:

Stage 1: a Parts-of-Speech (POS) tagging technique [35] is applied on the requirements that are defined using natural language.

Stage 2: the traceability links between the requirements and the models are generated through the *requirement-to-object-model* rules.

The *requirement-to-object-model* (RTOM) rules are specified by investigating grammatical patterns in requirements. Moreover, the RTOM rules are based on two kinds of relations between requirements and models. On one hand, *Overlap* relations are understood to be the relation between a sequence of terms in a requirement and a class, attribute, association, or association-end in a model. On the other hand, *Requires_Execution_Of* relations are understood to be the relation between a sequence of terms in a requirement and an operation in a model.

Figure 8 shows an example of both kinds of rules following the syntax that is defined in [10]. The top rule can establish an *Overlap* relation between a requirement and an attribute in a model. The bottom rule can establish a *Requires_Execution_Of* relation between a requirement and an operation in a

```
RTOM_RULE Rule-1:
EXISTS
   SEQUENCE(<x1/{NN1, NN2}>,<x2/{VBZ, VBR}>,<x3/{JJ}>) in Requirement;
   <x4/CLASS>, <x5/ATTRIBUTE> in Model
SUCH THAT
   ATTRIBUTE_OF(<x5>,<x4>) and CONTAINS(NAME(<x5>), <x3>) and (CONTAINS(NAME(<x4>), <x1>) or
   CONTAINS(NAME(<x4>), SINGULAR_FORM<x1>)
ACTION GENERATE
   OVERLAPS(Requirement, <x5>)
RTOM_RULE_END
```

```
RTOM_RULE Rule-2:
EXISTS
   SEQUENCE(<x1/{VV0,VVI,VVZ}>,<x2/{AT}>,<x3/{NN1, NN2}>) in Requirement;
   <x4/CLASS>, <x5/OPERATION> in Model
SUCH THAT
   OPERATION_OF(<x5>,<x4>) and MEMBER_OF(<x1>, SYNONYMS(STEREOTYPE(<x5>)) and
   CONTAINS(NAME(<x4>),<x3>)) or CONTAINS(NAME(<x4>), SINGULAR_FORM<x3>)
ACTION GENERATE
   REQUIRES_EXECUTION_OF(Requirement,<x4>)
RTOM_RULE_END
```

Figure 8: Example of *requirement-to-object-model* rules

model. These rules generate the traceability links between the requirement and the model presented in Figure 9.

The first rule in Figure 8 attempts to match a syntactic expression that consists of a noun (`<x1/{NN1, NN2}>`), the verb *to be* in the present form (`<x2/{VBZ, VBR}>`), and an adjective (`<x3/{JJ}>`) with an attribute in the model. The matching succeeds if: (a) the name of the attribute contains the adjective and the name of the class that defines the attribute contains the noun; or (b) the name of the attribute contains the adjective and the name of the class that defines the attribute contains the singular form of the noun. Therefore, in Figure 9, the sequence of terms `<NN1>button</NN1>` `<VBZ>is</VBZ>` `<JJ>pushed</JJ>` in the requirement and the attribute `Pushed` of the class `Button` satisfy the conditions of the rule. As a consequence, an *Overlap* relation is created between them.

The second rule in Figure 8 attempts to match a syntactic expression that consists of a verb (`<x1/{VV0,VVI,VVZ}>`), an article (`<x2/{AT}>`), and a noun (`<x3/{NN1, NN2}>`) with an operation in the model. The matching succeeds if: (a) the name of the operation contains the verb or is a synonym of the verb and the name of the class of the operation contains the noun; or (b) the name of the operation contains the verb or is a synonym of the verb and the name of the class of the operation contains the singular form of the noun. Therefore, in Figure 9, the sequence of terms `<VVI>open</VVI>` `<AT>the</AT>` `<NN1>door</NN1>` in the requirement and the operation `Set Open` of the class `Door` satisfy the conditions of the rule. As a conse-

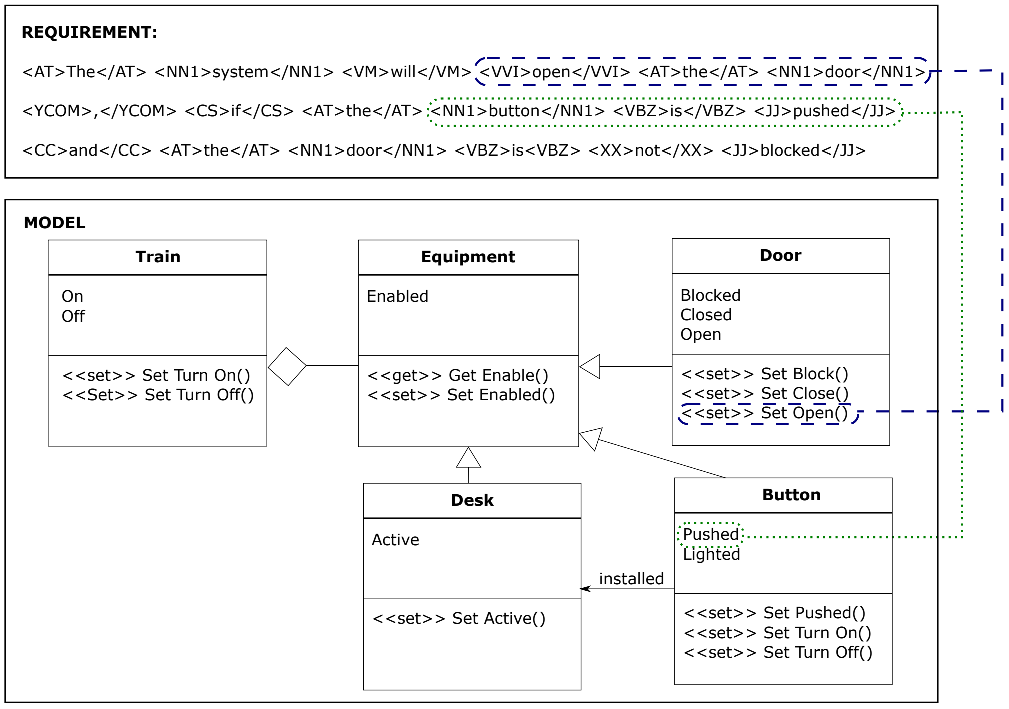


Figure 9: Example of traceability links generation based on RTOM rules

quence, a *Requires_Execution_Of* relation is created between them.

In [10], there are two different types of traceability rules: RTOM for traceability relations between requirements and model elements, and inter-requirement rules for traceability relations between different parts of a requirement statement. In total, the authors propose 26 rules for two domains: a software-intensive TV system created by Philips, and a university course management system. Since our approach is focused only on the traceability between requirements and model elements, this baseline only tackles the RTOM traceability rules for our domain. Therefore, based on the guides and the examples of rules that are provided by [10], a domain expert who was not involved in the research generated an initial set of rules for our domain. In addition, to mitigate the dependence on a single domain expert, a second expert who also was not involved in the research extended the set of rules. In the end, the extended set contains nine RTOM rules, which is similar to the number proposed by [10]. However, there is no significant difference between the results obtained using the initial set and the results obtained using the extended set. Specifically, the results described in this work correspond to the extended set, which are a bit better than those obtained from the initial set. Nonetheless, in both cases, the results are not as good as the ones obtained with our approach.

### *6.2.2. TLR-IR: Information Retrieval Baseline*

Information Retrieval (IR) [36, 37, 38] is a sub-field of computer science that deals with the automated storage and retrieval of documents. IR techniques have been successfully used to retrieve traceability links between different kinds of software artifacts in different contexts [39, 40, 41, 42, 43]. Specifically, in [11] and [12], De Lucia et al. use Latent Semantic Indexing (LSI) to recover traceability links between requirements and different kinds of software artifacts, including models in the form of use-case diagrams, among others. We use LSI to recover traceability links between requirements and models as one of the baselines for our work.

Specifically, given a certain requirement-model pair as input for LSI, we use the produced outcome of the technique to build a model fragment from the model that serves as a candidate for realizing the requirement. The following paragraphs provide more details on the process.

Latent Semantic Indexing (LSI) [44] is an automatic mathematical/statistical technique that analyzes relationships between *queries* and *documents* (bodies of text). Since both queries and documents are based on natural language, Natural Language Processing (NLP) techniques are used to process them. In fact, NLP has a direct and beneficial impact on the results, so before applying LSI, the queries and the documents are processed by a combination of NLP techniques defined in [23], which consists of tokenizing, lowercasing, removal of duplicate keywords, syntactical analysis, lemmatization, and stopword removal. Then, LSI constructs vector representations of both a user *query* and a corpus of text *documents* by encoding them as a *term-by-document co-occurrence matrix* and analyzes the relationships between those vectors to get a similarity ranking between the *query* and the *documents* (see Figure 10).

Figure 10 shows an example *term-by-document co-occurrence matrix*, with values associated to our case study, the vectors, and the resulting ranking. An overview of the elements of the matrix is provided in the following paragraphs:

- Each row in the matrix (*term*) stands for each of the words that compose the processed requirement and NL representation of the input model. The NL representation of the input model is extracted using the technique presented in [45]. For example, Figure 10 shows a set of representative words in the domain such as 'pantograph' or 'door' as the *terms* of each row.

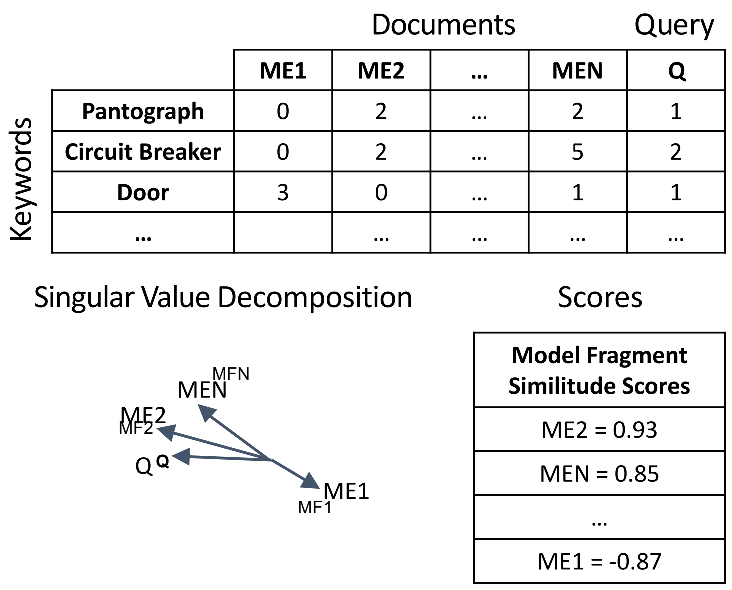
- Each column in the matrix (*document*) stands for one model element from one input model, taken from our real-world case study. For example, Figure 10 shows identifiers in the columns such as 'ME1' or 'ME2', which stand for the *documents* of those specific model elements.

- The final column stands for the *query*, which is a requirement in our case study.

- Each cell in the matrix contains the frequency with which the *term* of its row appears in the *document* denoted by its column. For instance, in Figure 10, the *term* 'pantograph' appears twice in the 'ME2' *document* and once in the *query*.

Vector representations of the *documents* and the *query* are obtained by normalizing and decomposing the *term-by-document co-occurrence matrix* using a matrix factorization technique called *Singular Value Decomposition* (SVD) [44]. SVD is a form of factor analysis, or more properly, it is the mathematical generalization of which factor analysis is a special case. In SVD, a rectangular matrix is decomposed into the product of three other matrices. One component matrix describes the original row entities as vectors of derived orthogonal factor values, a second one describes the original column entities in the same way, and the third one is a diagonal matrix containing scaling values such that when the three components are matrix-multiplied, the original matrix is reconstructed.

A three-dimensional graph of the SVD is provided in Figure 10. The graph shows the vectorial representations of some of the matrix columns. For legibility reasons, only a small set of the columns is represented. To measure the degree of similarity between vectors, the cosine between the *query* vector and the *documents* vectors is calculated. Cosine values that are closer to 1 denote a higher degree of similarity, and cosine values that are closer to -1 denote a lower degree of similarity. Similarity increases as vectors point in the same general direction (as more *terms* are shared between *documents*). With this measurement, the model elements are ordered according to their degree of similarity to the requirement.



| | ME1 | ME2 | ... | MEN | Q |
|---|---|---|---|---|---|
| **Pantograph** | 0 | 2 | ... | 2 | 1 |
| **Circuit Breaker** | 0 | 2 | ... | 5 | 2 |
| **Door** | 3 | 0 | ... | 1 | 1 |
| **...** | | ... | ... | ... | ... |

| **Model Fragment Similitude Scores** |
|---|
| ME2 = 0.93 |
| MEN = 0.85 |
| ... |
| ME1 = -0.87 |

Figure 10: Example of Traceability Link Recovery using Latent Semantic Indexing

The relevancy ranking (which can be seen in Figure 10) is produced according to the degrees of similarity calculated. In this example, LSI retrieves 'ME2' and 'MEN' in the first and second position of the relevancy ranking since the *query-documents* cosines are '0.9343' and '0.8524', implying a high degree of similarity between the model elements and the requirement. In contrast, the 'M1' model element is returned to a lower position in the ranking since its *query-document* cosine is '-0.8736', implying a lower degree of similarity.

From the ranking, of all the model elements, only those model elements that have a degree of similarity greater than $x$ must be taken into account. A good heuristic that is widely used is $x = 0.7$. This value corresponds to a $45°$ angle between the corresponding vectors. Even though the selection of the threshold is an issue under study, the heuristic chosen for this work has yielded good results in other similar works [46, 47].

Following this principle, the elements with a degree of similarity equal or greater than to $x = 0.7$ are taken to conform a model fragment, which is a candidate for realizing the requirement. In the example provided in Figure 10, ME2 and MEN are model elements that conform part of the model fragment that is obtained by this baseline for the requirement because their cosine values are greater than the 0.7 threshold. The model fragment

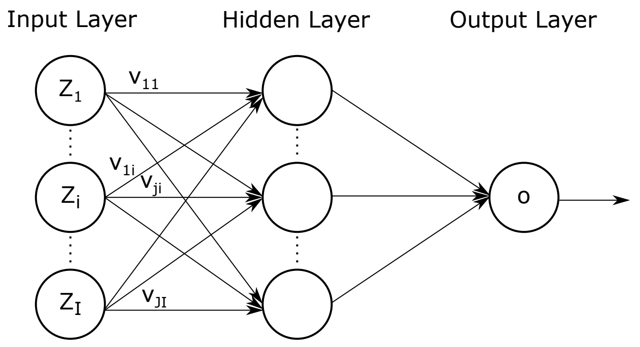


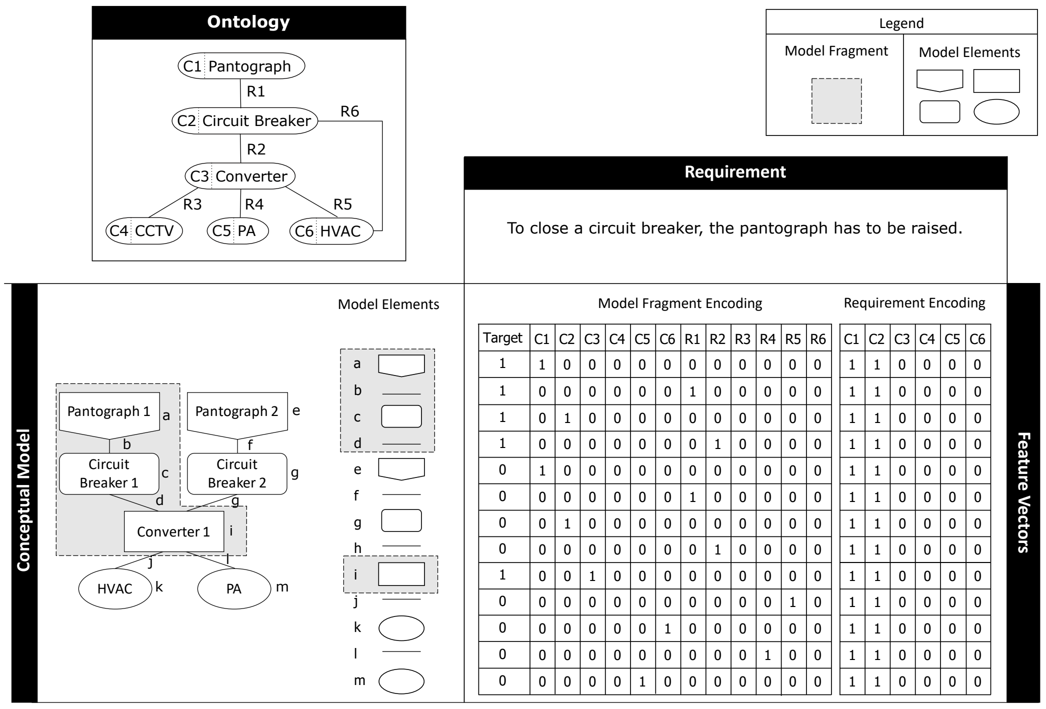
Figure 11: Feedforward Neural Network

generated in this manner is the final output of the TLR-IR baseline.

*6.2.3. TLR-FNN: Feedforward Neural Network Baseline*

Feedforward Neural Networks (FNNs) represent a traditional neural network structure and lay the foundation for many other structures [48]. Data flow always moves one direction, from input layer to hidden layer, then to output layer; it never goes backwards. Figure 11 shows the structure of a FNN where the FNN receives a vector of $I$ input signals, $z = (z_1, z_2, ..., z_I)$. The neurons of the hidden layer assign to each input signal, $Z_i$, its respective weight, $v_i$, to strengthen or deplete the input signal. Weighted inputs are accumulated at each neuron and then an activation function determines the output (or firing strength) of each neuron, $o$. In fact, the strength of the output is further influenced by a threshold value, which is also referred to as the bias; thus, the activation function receives both the input signal and the bias to determine the output of each neuron [13].

While Figure 11 shows only one hidden layer, a FNN can have more than one hidden layer. However, it has been proved that FNNs with monotonically increasing differentiable functions can approximate any continuous function with one layer, provided that the hidden layer has enough hidden neurons [49]. Specifically, the network architecture of the FNN implemented here is a dense layer that is followed by the final softmax layer. Moreover, we performed a hyperparameter optimization based on the random search optimization provided by the Deep Learning for Java library. For all of the layers, the hyperparameter optimization resulted in an initial learning rate of 0.0035, and the Gaussian distribution recommended in [50] for weight initialization. In addition, for the dense layer, the hyperparameter optimization resulted in



To close a circuit breaker, the pantograph has to be raised.

| | Model Fragment Encoding | | | | | | | | | | | | Requirement Encoding | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Target | C1 | C2 | C3 | C4 | C5 | C6 | R1 | R2 | R3 | R4 | R5 | R6 | C1 | C2 | C3 | C4 | C5 | C6 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |

Figure 12: Example of encoding of the knowledge base at the model-level

a layer size of 128 and the randomized rectified linear unit (RRELU) as the activation function.

In addition, since FNN is a typical kind of supervised machine learning method, the training process is required to adjust weights and bias for inputs. Therefore, a training set has to be used to train the FNN and to determine how well the FNN has learned [51]. Moreover, the testing process is required to recover the traceability link between a model and a requirement. However, both the training set and the testing are a bit different from the sets used in TLR-ELtoR. Since this baseline is not based on a EA, it cannot generate and evolve model fragments. Therefore, this baseline works at the model-element level instead of at the model-fragment level. For this reason, both the training set and the testing set consist of a set of vectors, where each vector represents the relation between a model element and a requirement.

Figure 12 shows an example of the encoding of the knowledge base at the model-element level. From a sample of the knowledge base, each element of the model is encoded by means of a feature vector. Then, the encoding for the requirement is included in all of the feature vectors. Finally, a numerical value is included as target for the training. This value is equal to 0, if the model element is not present in the model fragment. Otherwise, this value is equal to 1, if the model element is present in the model fragment. This value is used by the FNN to learn relations between the model elements and the requirements. In summary, each sample of the knowledge base is encoded using several feature vectors, one for each element in the model.

Likewise, the model and the requirement for the testing set are encoded at the model-element level. Each model element leads to one feature vector, and the encoding of the requirement is included in all of the feature vectors.

In addition, Figure 12 also shows a limitation of the encoding proposed in [22] when it is applied at the model-element level. Several feature vectors contain the same feature values, but different target values (e.g., both pantographs). Taking into account the ontology in Figure 12, there is no way to differentiate between two elements of the same type; for example, both pantographs have the same encoding. However, the ontology can be extended to tackle specific properties of each type of element. For example, the ontology of Figure 12 can be extended to include the status of each pantograph as an attribute of the pantograph concept. Therefore, the two pantographs can be differentiated taking into account if their status is in the up, down, or middle position. Specifically, in our case study, the ontology was extended with 14 attributes that empowered us to mitigate this threat.

### *6.2.4. TLR-RNN: Recurrent Neural Network Baseline*

Since the number of parameters in a fully connected FNN can grow extremely large as the width and depth of the network increases, researchers have proposed other neural network structures targeting different types of practical problems. Recurrent Neural Networks (RNNs) are particularly well suited for processing sequential data such as text and audio. While FNNs have no feedback connections to previous layers, RNNs have these feedback connections to model the temporal characteristics of the problem being learned [13]. Moreover, RNNs have successfully been applied in TLR in some recent works [9].

Although RNNs are specifically designed to process sequential data, RNNs have showed great results in some cases of non-sequential input information, for instance, image captioning [52] or prediction of hospital readmission [53]. In these works, even if the input data is not in the form of sequences, they can make classifiers able to learn so that they process data in sequential order only [53]. In our case, even if the models are not sequential data, we can order the feature vectors of the model elements so that a classifier trained by a RNN benefits from the sequential order of the model elements. For example, taking into account the Figure 12, the three first feature vectors match the model elements: *Pantograph1*, the relation *Pantograph1-Circuit Breaker1*, and *Circuit Breaker1*. Taking into account their order, even if we knew that *Pantograph1* is related to the requirement, we could not determine if the other two model elements are related to the requirement. However, if we knew that the *Pantograph1* and the relation *Pantograph1-Circuit Breaker1* are related to the requirement, it would be certainly reasonable to assume that the *Circuit Breaker1* is related to the requirement.

Figure 13 shows the structure of an Elman RNN, which is a RNN based on the extension of a FNN. As illustrated in Figure 13, data flow moves from an input layer to a hidden layer, but there is a new layer, named context layer, that makes a copy of the hidden layer. This context layer serves as an extension of the input layer, feeding signals that represent previous network states to the hidden layer. Therefore, the input vector is $z = (z_1, ..., z_I I, z_{I+1}, ..., z_{I+J})$, where the first $I$ signals are the actual inputs of the network and the $J$ signals are the context units [13].

A prominent drawback of the standard RNN model is that the network degrades when long dependencies exist in the sequence due to the phenomenon of exploding or vanishing gradients during back-propagation [54]. This makes a standard RNN model difficult to train. The exploding gradient problem can be effectively addressed by scaling down the gradient when its norm is bigger than a preset value (i.e., Gradient Clipping) [54]. To address the vanishing gradient problem of the standard RNN model, the RNN network that is used as baseline in this work applies Long Short Term Memory (LSTM), which is a variant provided by researchers that has mechanics to preserve long-term dependencies [9].

LSTM networks include a memory cell vector in the recurrent neuron to preserve long-term dependencies [55]. LSTM also introduces a gating mechanism to control when and how to read or write information to the memory cell. A gate in LSTM usually uses a sigmoid function $\sigma(z) = 1/(1 + e - z)$ and controls information throughput using a point-wise multiplication operation. Specifically, when the sigmoid function outputs 0, the gate forbids any information from passing, while all information is allowed to pass when the sigmoid function output is 1 [9]. Each LSTM neuron contains an input

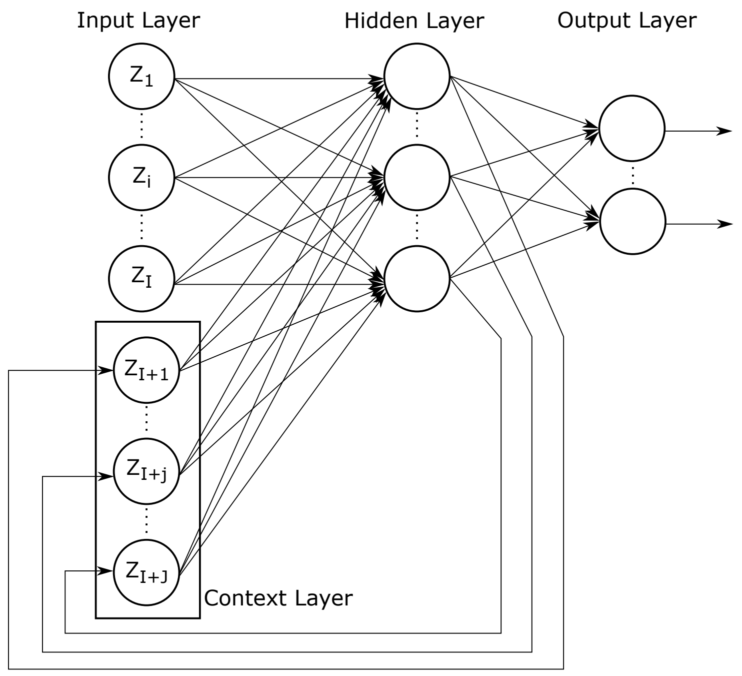


Figure 13: Elman Simple Recurrent Neural Network

gate, a forget gate, and an output gate. The input gate controls how much each signal in a candidate vector should be "remembered". The forget gate controls how much each signal in the previous memory neuron state should be retained, so the neuron "remembers" information until it is erased by the forget gate. Finally, the output gate controls when a signal output is used in the activation function [9].

Specifically, the network architecture of the RNN implemented is a LSTM layer followed by the final softmax layer. Moreover, we performed a hyperparameter optimization based on the random search optimization provided by the Deep Learning for Java library. For all of the layers, the hyperparameter optimization resulted in an initial learning rate of 0.02 and the Normal distribution described in [56] for the weight initialization. In addition, for the LSTM layer, the hyperparameter optimization resulted in a layer size of 223 and the standard sigmoid activation function as the activation function.

Since RNN is also based on supervised learning such as the TLR-FNN baseline, training and testing processes are also required. Therefore, we have performed the same encoding as the TLR-FNN baseline (See Figure 12).

### *6.2.5. TLR-LtoR: Learning to Rank Baseline*

Taking into account this baseline, we want to determine if the better results of TLR-ELtoR are due to the combination of the evolutionary algorithm and LtoR, or there is no need to combine the two to get these results. For this purpose, this baseline is based only on LtoR, and the model fragments that are used as input for the LtoR process are generated randomly through a standard random search.

We used this algorithm as outlined in Algorithm 2 (available in [57]). The algorithm starts with a random initial model fragment, as the best fragment. A new random model fragment is then generated and assessed using LtoR. Then, the values provided by LtoR for both fragments, the best one and the new one, are compared and the model fragment with the greatest value is selected as the best one. The search then goes back to the second step, generating and assessing a new model fragment, and this loop is repeated until a stop condition is met.

Therefore, this baseline does not take advantage of evolving model fragments to guide the exploration of the models, as our approach does thanks to the evolutionary algorithm. Since there is no a guide to explore the models, the search for the model fragment that realizes a specific requirement is performed by brute-force.

Since TLR-LtoR is also based on LtoR such as TLR-ELtoR, training and testing are also required. Therefore, the same steps that were described for TLR-ELtoR in Section 5 are applied in TLR-LtoR to encode model fragments as feature vectors, to train a classifier from the knowledge base, and to test the test cases. In addition, the technical details, such as the LtoR algorithm and the cross-validation method, are also the same ones defined in the setup of TLR-ELtoR (See subsection 6.1.1). However, the stop condition is different from TLR-ELtoR in order so that the comparison between them is fair.

The stop condition in TLR-ELtoR was set up to perform 200 iterations of the evolutionary algorithm, where each iteration evaluated 120 model fragments. Therefore, for each test case, the approach evaluated a total of 24000 model fragments. However, TLR-LtoR approach only evaluates one model fragment for each iteration, so the stop condition was set up to perform 24000 iterations in order to evaluate the same number of model fragments.

## *6.3. Case Study*

The case study where we applied our approach was CAF, a worldwide provider of railway solutions. Their trains can be found all over the world and in different forms (regular trains, subway, light rail, monorail, etc.). A train unit is furnished with multiple pieces of equipment in its vehicles and cabins. These pieces of equipment are often designed and manufactured by different providers, and their aim is to carry out specific tasks for the train. Some examples of these devices are: the traction equipment, the compressors that feed the brakes, the pantograph that harvests power from the overhead wires, and the circuit breaker that isolates or connects the electrical circuits of the train. The control software of the train unit is in charge of making all of the equipment cooperate in order to achieve the train functionality, while guaranteeing compliance with the specific regulations of each country. The following video illustrates the CAF models: `youtube.com/watch?v=Ypcl2evEQB8`

Our evaluation includes 20 test cases, which are composed of a requirement, a product model, a knowledge base, and an ontology. A detailed description of each of them and how they are used in our approach TLR-ELtoR is provided below:

- The **requirements** have about 25 words. In TLR-ELtoR, these requirements and the models are used to generate the testing sets.
- The **models** have about 650 elements. In TLR-ELtoR, model fragments are generated by the evolutionary algorithm through genetic operations from these models. Then, these model fragments and the requirements compose the testing set.
- The **knowledge base** includes 103 samples. Specifically, each of these samples contains a requirement, a model fragment that has about 15 elements, and an assessment. In TLR-ELtoR, the entire knowledge base composes the training set that is used to train the classifier. In the end, the number of samples in the knowledge base allows the generation of a suitable, but not perfect, classifier [58]. Figure 14 shows the distribution of scores in the knowledge base, taking into account how many samples of the knowledge base are in each scores range.
- The **ontology** contains a total of 54 elements between concepts and relations. In TLR-ELtoR, the ontology is used to encode the knowledge base in the training process, and to encode the requirements and the models in the testing process.

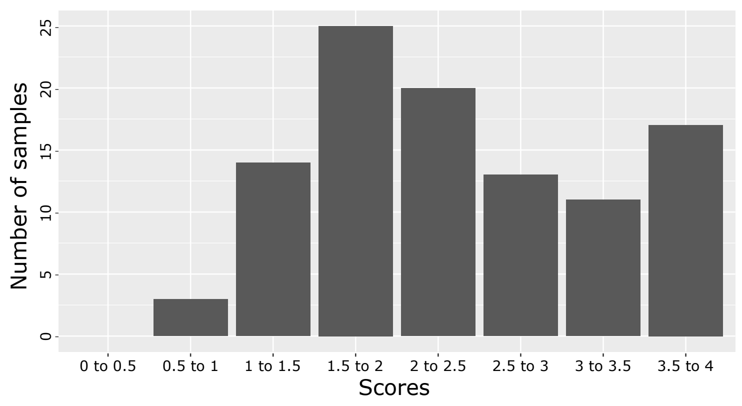


Figure 14: Distribution of scores in the knowledge base

For each test case, we followed the experimental setup described in Figure 7. Each test case was run 30 times. As suggested by [59], given the stochastic nature of the TLR-ELtoR approach, several repetitions are needed to obtain reliable results. Finally, the results were evaluated and compared to the oracle. The oracle contains the **approved traceability**, which consists of a set of model fragments, where each model fragment contains the model elements that are required by the requirement. In other words, the oracle contains the solutions for each test case, so the oracle had 20 model fragments, one for each test case.

## *6.4. Implementation details*

We used the Eclipse Modeling Framework to manipulate the models and CVL to manage the model fragments. For the development of the TLR-Linguistic baseline, the Stanford POS Tagger [60] was utilized. The LSI technique used within the TLR-IR baseline was implemented using the Efficient Java Matrix Library (EJML [61]). The neural networks in TLR-FNN and TLR-RNN were developed and tuned by means of the Deep Learning for Java library [62]. The genetic operations were built upon the Watchmaker Framework for Evolutionary Computation [63]. Finally, RankBoost was implemented using the RankLib library [64].

## *6.5. Results*

In Table 1, we outline the results, which are aggregated for each of the baselines and for our approach. Each row shows the Precision, Recall, F-measure, and MCC obtained through each technique.

Table 1: Mean Values and Standard Deviations for Precision, Recall, F-Measure, and Matthews Correlation Coefficient (MCC) for the baselines and the TLR-ELtoR approach

| | Precision | Recall | F-Measure | MCC |
|---|---|---|---|---|
| TLR-Linguistic | 37.38 ± 16.18 | 48.61 ± 19.78 | 40.41 ± 16.19 | 0.40 |
| TLR-IR | 18.09 ± 25.55 | 53.45 ± 38.70 | 21.69 ± 23.95 | 0.21 |
| TLR-FNN | 8.20 ± 0.10 | 100 ± 0.00 | 14.06 ± 0.14 | -0.84 |
| TLR-RNN | 8.37 ± 0.09 | 100 ± 0.00 | 14.34 ± 0.14 | -0.77 |
| TLR-LtoR | 13.01 ± 26.08 | 11.85 ± 18.24 | 10.27 ± 17.57 | 0.07 |
| TLR-ELtoR | 59.91 ± 33.39 | 78.95 ± 15.16 | 62.50 ± 27.76 | 0.64 |

As the table shows, TLR-ELtoR achieves the best results for most performance indicators, providing a mean precision value of 59.91%, a recall value of 78.95%, a combined F-measure value of 62.50%, and a MCC value of 0.64. In contrast, the TLR-Linguistic baseline, the TLR-IR baseline, and the TLR-LtoR baseline present worse results in all of the measurements: the TLR-Linguistic baseline attains 37.38% precision, 48.61% recall, 40.41% F-measure, and 0.40 MCC; the TLR-IR baseline achieves 18.09% precision, 53.45% recall, 21.69% F-measure, and 0.21 MCC; and the TLR-LtoR baseline attains 13.01% precision, 11.85% recall, 10.27% F-measure, and 0.07 MCC. On the other hand, both the TLR-FNN baseline and the TLR-RNN baseline achieve the best results for recall, but they present the worst results for the rest of the indicators: the TLR-FNN attains 8.20% precision, 100% recall, 14.06% F-measure, and -0.84 MCC; and the TLR-RNN baseline achieves 8.37% precision, 100% recall, 14.34% F-measure, and -0.77 MCC.

## 7. Statistical Analysis

To properly compare the different configurations, the data resulting from the empirical analysis was analyzed using statistical methods.

### *7.1. Statistical Significance*

A statistical test must be run to assess whether there is enough empirical evidence to claim that there is a difference between two approaches (e.g., A is better than B). To achieve this, two hypotheses are defined: the null

Table 2: Quade test statistic and $p - Values$

| | Recall | Precision |
|---|---|---|
| p-Value | $2.20 \times 10^{-16}$ | $1.7 \times 10^{-10}$ |
| Statistic | 35.27 | 14.41 |

hypothesis $H_0$, and the alternative hypothesis $H_1$. The null hypothesis $H_0$ is typically defined to state that there is no difference between the approaches, whereas the alternative hypothesis $H_1$ states that the configurations differ. In such a case, a statistical test aims to verify whether the null hypothesis $H_0$ should be rejected.

Statistical tests provide a probability value, $p - Value$. The $p - Value$ obtains values between 0 and 1. The lower the $p - Value$ of a test, the more likely that the null hypothesis is false. It is accepted by the research community that a $p - Value$ under 0.05 is statistically significant [65], and so the hypothesis $H_0$ can be considered false.

The test carried out depends on the properties of the data. Since our data does not follow a normal distribution in general, our analysis required the use of nonparametric techniques. There are several tests for analyzing this kind of data; however, the Quade test is the most powerful one when working with real data [66]. In addition, according to Conover [67], the Quade test is the one that has shown the best results for a low number of approaches (no more than 4 or 5 approaches).

Table 2 shows the Quade test statistic and $p - Values$ for recall and precision. Since the $p - Values$ are smaller than 0.05, we rejected the null hypothesis. Consequently, we can state that there are differences among the five approaches.

Nevertheles, with the Quade test, we cannot answer the following question: *Which of the approaches gives the best performance?* In this case, the performance of each approach should be individually compared with all of the other alternatives. In order to do this, we performed an additional post hoc analysis. This kind of analysis performs a pair-wise comparison among the results of each approach, determining whether statistically significant differences exist among the results of a specific pair of approaches.

Table 3 shows the $p - Values$ of Holm's post hoc analysis for each specific pair of approaches. Almost of all the $p - Values$ shown in this table are smaller than 0.05, except for some cases: the recall comparison between

Table 3: Holm's Post Hoc $p - Values$

| | Recall | Precision |
|---|---|---|
| TLR-Linguistic vs TLR-IR | 0.49 | $6.1 \times 10^{-04}$ |
| TLR-Linguistic vs TLR-FNN | $1.0 \times 10^{-07}$ | $3.4 \times 10^{-07}$ |
| TLR-Linguistic vs TLR-RNN | $1.0 \times 10^{-07}$ | $3.4 \times 10^{-07}$ |
| TLR-Linguistic vs TLR-LtoR | $1.9 \times 10^{-05}$ | $2.5 \times 10^{-03}$ |
| TLR-Linguistic vs TLR-ELtoR | $3.3 \times 10^{-06}$ | $8.4 \times 10^{-03}$ |
| TLR-IR vs TLR-FNN | $4.2 \times 10^{-06}$ | 0.04 |
| TLR-IR vs TLR-RNN | $4.2 \times 10^{-06}$ | 0.04 |
| TLR-IR vs TLR-LtoR | $1.1 \times 10^{-03}$ | 0.27 |
| TLR-IR vs TLR-ELtoR | 0.041 | $7.3 \times 10^{-05}$ |
| TLR-FNN vs TLR-RNN | 0.0 | $1.1 \times 10^{-03}$ |
| TLR-FNN vs TLR-LtoR | $2.9 \times 10^{-08}$ | 0.97 |
| TLR-FNN vs TLR-ELtoR | $8.3 \times 10^{-06}$ | $1.9 \times 10^{-07}$ |
| TLR-RNN vs TLR-LtoR | $2.9 \times 10^{-08}$ | 0.97 |
| TLR-RNN vs TLR-ELtoR | $8.3 \times 10^{-06}$ | $1.9 \times 10^{-07}$ |
| TLR-LtoR vs TLR-ELtoR | $2.7 \times 10^{-07}$ | $3.8 \times 10^{-06}$ |

TLR-Linguistic and TLR-IR, the recall comparison between TLR-FNN and TLR-RNN, the precision comparison between TLR-IR and TLR-LtoR, the precision comparison between TLR-FNN and TLR-LtoR, and the precision comparison between TLR-RNN and TLR-LtoR. Therefore, significant differences for one of the performance measurements were obtained in all of the comparisons.

### *7.2. Effect Size*

Statistically significant differences can be obtained even if they are so small as to be of no practical value [65]. It is then important to assess whether an approach is statistically better than another and to assess the magnitude of the improvement. *Effect size* measures are needed to analyze this.

Table 4: $\hat{A}_{12}$ statistic for each pair of approaches

| | Recall | Precision |
|---|---|---|
| TLR-Linguistic vs TLR-IR | 0.45 | 0.81 |
| TLR-Linguistic vs TLR-FNN | 0.0 | 0.93 |
| TLR-Linguistic vs TLR-RNN | 0.0 | 0.93 |
| TLR-Linguistic vs TLR-LtoR | 0.91 | 0.86 |
| TLR-Linguistic vs TLR-ELtoR | 0.17 | 0.30 |
| TLR-IR vs TLR-FNN | 0.13 | 0.55 |
| TLR-IR vs TLR-RNN | 0.13 | 0.54 |
| TLR-IR vs TLR-LtoR | 0.78 | 0.65 |
| TLR-IR vs TLR-ELtoR | 0.32 | 0.15 |
| TLR-FNN vs TLR-RNN | 0.5 | 0.48 |
| TLR-FNN vs TLR-LtoR | 1 | 0.67 |
| TLR-FNN vs TLR-ELtoR | 0.85 | 0.07 |
| TLR-RNN vs TLR-LtoR | 1 | 0.68 |
| TLR-RNN vs TLR-ELtoR | 0.85 | 0.07 |
| TLR-LtoR vs TLR-ELtoR | 0.03 | 0.11 |

For a non-parametric effect size measure, we used Vargha and Delaney's $\hat{A}_{12}$ [68]. $\hat{A}_{12}$ measures the probability that running one approach yields higher values than running another approach. If the two approaches are equivalent, then $\hat{A}_{12}$ will be 0.5.

For example, $\hat{A}_{12} = 0.7$ means that we would obtain better results in 70% of the runs with the first of the pair of approaches that have been compared, and $\hat{A}_{12} = 0.3$ means that we would obtain better results in 70% of the runs with the second of the pair of approaches that have been compared. Thus, we have an $\hat{A}_{12}$ value for every pair of approaches.

Table 4 shows the values of the effect size statistics between every pair of approaches.

**TLR-Linguistic vs TLR-IR:** The $\hat{A}_{12}$ measure value indicates that, of the two approaches, TLR-Linguistic will obtain better results in 58% of the cases for recall, while TLR-IR will obtain better precision values in 81% of the cases.

**TLR-Linguistic vs TLR-FNN:** The $\hat{A}_{12}$ measure value indicates that, of the two approaches, TLR-FNN will obtain better results in 100% of the cases for recall, while TLR-Linguistic will obtain better precision values in 93% of the cases.

**TLR-Linguistic vs TLR-RNN:** The $\hat{A}_{12}$ measure value indicates that, of the two approaches, TLR-RNN will obtain better results in 100% of the cases for recall, while TLR-Linguistic will obtain better precision values in 93% of the cases.

**TLR-Linguistic vs TLR-LtoR:** The $\hat{A}_{12}$ measure value indicates that TLR-Linguistic will obtain better results than TLR-LtoR in 91% of the cases for recall, and better precision values in 86% of the cases.

**TLR-Linguistic vs TLR-ELtoR:** The $\hat{A}_{12}$ measure value indicates that TLR-ELtoR will obtain better results than TLR-Linguistic in 83% of the cases for recall, and better precision values in 70% of the cases.

**TLR-IR vs TLR-FNN:** The $\hat{A}_{12}$ measure value indicates that, of the two approaches, TLR-RNN will obtain better results in 87% of the cases for recall, while TLR-IR will obtain better precision values in 55% of the cases.

**TLR-IR vs TLR-RNN:** The $\hat{A}_{12}$ measure value indicates that, of the two approaches, TLR-RNN will obtain better results in 87% of the cases for recall, while TLR-IR will obtain better precision values in 54% of the cases.

**TLR-IR vs TLR-LtoR:** The $\hat{A}_{12}$ measure value indicates that, of the two approaches, TLR-IR will obtain better results than TLR-LtoR in 78% of the cases for recall, and better precision values in 65% of the cases.

**TLR-IR vs TLR-ELtoR:** The $\hat{A}_{12}$ measure value indicates that, of the two approaches, TLR-ELtoR will obtain better results than TLR-IR in 68% of the cases for recall, and better precision values in 85% of the cases.

**TLR-FNN vs TLR-RNN:** The $\hat{A}_{12}$ measure value indicates that, of the two approaches, TLR-RNN will obtain better results in 50% of the cases for recall, and better precision values in 52% of the cases.

**TLR-FNN vs TLR-LtoR:** The $\hat{A}_{12}$ measure value indicates that, of the two approaches, TLR-FNN will obtain better results than TLR-LtoR in 100% of the cases for recall,and better precision values in 68% of the cases.

**TLR-FNN vs TLR-ELtoR:** The $\hat{A}_{12}$ measure value indicates that, of the two approaches, TLR-FNN will obtain better results in 85% of the cases for recall, while TLR-ELtoR will obtain better precision values in 93% of the cases.

**TLR-RNN vs TLR-LtoR:** The $\hat{A}_{12}$ measure value indicates that, of the two approaches, TLR-RNN will obtain better results than TLR-LtoR in 100% of the cases for recall,and better precision values in 68% of the cases.

**TLR-RNN vs TLR-ELtoR:** The $\hat{A}_{12}$ measure value indicates that, of the two approaches, TLR-RNN will obtain better results in 85% of the cases for recall, while TLR-ELtoR will obtain better precision values in 93% of the cases.

**TLR-LtoR vs TLR-ELtoR:** The $\hat{A}_{12}$ measure value indicates that, of the two approaches, TLR-ELtoR will obtain better results than TLR-LtoR in 97% of the cases for recall,and better precision values in 89% of the cases.

The obtained $\hat{A}_{12}$ values show that TLR-ELtoR is superior to all of the baselines for precision. Moreover, TLR-ELtoR is also superior to TLR-Linguistic, TLR-IR, and TLR-LtoR on recall, meaning that TLR-ELtoR will obtain better results than these three approaches in most of the cases. Overall, these measurements confirm that, for recall and precision, TLR-ELtoR outperforms the baselines (TLR-IR and TLR-Linguistic) that obtain the best results for TLR between requirements and models. Moreover, these measurements confirm that, for precision, TLR-ELtoR outperforms the ML baselines (TLR-FNN and TLR-FNN) that have successfully been applied recently in TLR. Finally, these measurements confirm that, for recall and

Table 5: Required artifacts for each approach

| | | Approaches | | | | | |
|---|---|---|---|---|---|---|---|
| | | TLR-Linguistic | TLR-IR | TLR-FNN | TLR-RNN | TLR-LtoR | TLR-ELtoR |
| Artifacts | Models | X | X | X | X | X | X |
| | Requirements | X | X | X | X | X | X |
| | Knowledge Base | | | X | X | X | X |
| | Ontology | | | X | X | X | X |
| | Rules | X | | | | | |

precision, TLR-ELtoR outperforms the baseline (TLR-LtoR) that explores the search space by means of brute-force.

## 8. Discussion

In this section, we discuss what prerequisites are needed by each approach, what properties affect the results and limit the approaches. We also discuss why TLR-ELtoR is less sensitive to tacit knowledge and vocabulary mismatch than the baselines. These advantages lead to the better results of TLR-ELtoR.

### *8.1. Prerequisites and Properties*

Both our approach and the approaches in the baselines need some prerequisites to be applied. If one of their prerequisites is not satisfied, the approach would not be used in that domain. Table 5 shows what artifacts are needed to apply each approach.

Table 5 shows that all of the approaches need models and requirements. Specifically, the models where requirements have to be located must conform to MOF (the OMG metalanguage for defining modeling languages) and that requirements must be provided using natural language. Moreover, all of the approaches that are based on Machine Learning (TLR-ELtoR, TLR-LtoR, TLR-FNN, and TLR-RNN) need a knowledge base to train and an ontology to encode the models and requirements. Specifically, the knowledge base must be composed of a set of feature vectors with the format described in [69], and the ontology must contain a set of concepts and the relations with each other (See Figure 6). Finally, the TLR-Linguistic approach needs rules to identify relations between model elements and requirement words. The rules have to be defined following the guides and examples in [10].

Table 6: Artifacts whose properties have an impact on the results

| | | Properties | | | | |
|---|---|---|---|---|---|---|
| | | Homogeneity | Completeness | Heterogeneity | Size | Volume |
| Artifacts | Models | X | | | X | |
| | Requirements | X | X | | | |
| | Knowledge Base | | X | X | X | |
| | Ontology | | X | | X | |
| | Rules | | X | | | X |

Therefore, even though the training in TLR-ELtoR is beneficial in avoiding to a large extent issues such as tacit knowledge and vocabulary mismatch, it is necessary to have access to a knowledge base and an ontology to perform the training. In industrial domains, especially long-living ones, where requirements and models have been stored for years, a knowledge base may be easily available. Also, thanks to the wide experience of the employees in companies of this kind, the main concepts and relations could be identified by experts in the domain. However, in other scenarios, such as when only the first product has been developed, TLR-ELtoR cannot be applied.

In addition, even though we had of all the necessary artifacts to apply our approach, the results may not be as good as possible. In fact, some properties of the artifacts have an impact on the results. For example, if there is not enough information in the knowledge base, TLR-ELtoR would not train properly, so the results would be worse than expected. Table 6 shows the properties that we have identified in this work and that had an impact on the obtained results.

The following paragraphs provide more details about the properties identified in Table 6:

- **Models** may be developed by several engineers and at different times, so the terms used to describe model elements may be different (e.g., pantograph and panto are two different terms used in our models to refer to the same concept: *pantograph*). Therefore, the first model property that affects to the results is homogeneity. The second one is the size, which has an impact on the result based on the understanding of the models [70].

- **Requirements** may be defined by different engineers and at different times, so the homogeneity of the requirements, like the homogeneity of the models, has an impact on the results. The results are also affected by the completeness of the requirements. Often, when requirements are written, part of the domain knowledge related to the requirements is not embodied in them because tacit knowledge about the domain is assumed to be known by all of the domain experts. Therefore, the requirements are more or less complete in accordance with how many assumptions are made by the engineers. In the end, requirements may lose part of the information that is required because of these assumptions.

- **Knowledge Base** contains the information necessary to train the classifier, so this information must be enough to train the classifier. If the knowledge base only contains the information to recover the traces between one requirement and one model, the classifier may not learn how to recover the traces for other requirements or models. Therefore, including heterogeneity samples of traces in the knowledge base provides more complete information for the training. In addition, some Machine Learning techniques require a larger knowledge base than others to provide suitable results, so the technique must be selected based on the available knowledge base.

- **Ontology** is composed of the main concepts and relations of a domain. Therefore, if a relevant concept or relation is not present in the ontology, the encoding for the fitness function will not take it into account and the training may be incomplete, leading to worse results. For this reason, the first property to keep in mind for the ontology is completeness. Moreover, if the ontology contains unnecessary concepts or relations, the number of features for the encoding would be greater and a great number of features in the training step leads to overfitting. Therefore, we must also take into account the size of the ontology.

- **Rules** are defined by humans through the manual comparison of the models and requirements. Therefore, the completeness of the rules depends on how well engineers understand the models and requirements and how complex these models and requirements are. The volume of rules also affects the results. If only one rule is defined, the approach only recovers one type of model element, so the approach may need several rules. However, a large number of rules does not guarantee the best results. Therefore, both completeness and volume must be taken in account.

## *8.2. Advantages of TLR-ELtoR*

This section discusses why TLR-ELtoR achieves better results than the baselines regarding three aspects: tacit knowledge, vocabulary mismatch, and available documentation.

### *8.2.1. Tacit Knowledge*

Often, when requirements are written, part of the domain knowledge related to the requirements is not embodied in them. The tacit knowledge about the domain is assumed to be known by all of the domain experts, so it is never formalized in writing. This behavior has been reported in previous works [71, 72]. For example, given the requirement: *At all stations, the doors are automatically opened*, the engineers understand that the doors have to be opened in all of the stations, without being requested by a passenger. However, this requirement also embodies tacit knowledge that is not written but is obvious to the domain engineers: *The train has doors on both sides, but only the doors on the side of the platform will be opened, while the doors on the side of the tracks will remain closed*, and *all of the doors on one side will be opened, except the driver's door in the cabin.*

The tacit knowledge is not reflected in the text of the requirements. This tacit knowledge is shared among the engineers that write the requirements and the engineers that read the requirements. Therefore, both the text of the requirements and tacit knowledge are used to build the models. As a result, the model contains elements that are related to text of the requirement, but the model also contains elements that are related to the tacit knowledge. However, since part of the knowledge is not reflected in the text of the requirement, recovering the most relevant model fragment for a requirement is complex.

Both TLR-IR and TLR-Linguistic depend, to a large extent, on the text of the requirement. TLR-IR evaluates the similarity between the requirement and the model fragment according to the co-occurrences of terms between the two. TLR-Linguistic evaluates the similarity between the requirement and the model fragment according to patterns that relate the terms in the requirement with the elements in the model fragment. In both cases, the lack of terms that is caused by the tacit knowledge makes it impossible to locate the elements from the model that are relevant to the requirement.

In contrast, TLR-ELtoR is less sensitive to tacit knowledge due to training. In the training, the requirements of the knowledge base are linked to the model fragments of the knowledge base. Even though the text of requirements is inaccurate due to tacit knowledge, the linked model fragments are complete. Consequently, the classifier is not only trained from the text of the requirements, but also from the elements of the model fragments. Therefore, the classifier learns that certain elements of models are relevant to certain requirements even though these elements are not described properly in the text of the requirements. As TLR-ELtoR depends, to a lesser extent, on the text of the requirement than TLR-IR and TLR-Linguistic, when the requirements have a lack of terms due to tacit knowledge, the results that are obtained through TLR-ELtoR are better than the results obtained through TLR-IR and TLR-Linguistic.

### *8.2.2. Vocabulary Mismatch*

Vocabulary mismatch is caused by the use of different terms to reference the same concept in the requirement and the model. In industrial environments, sometimes the engineer who is in charge of writing the requirement is not the same engineer assigned to building the model. Moreover, both the requirement and the model may be manipulated by different engineers.

Even though TLR-IR, TLR-Linguistic, and even TLR-ELtoR, may use Natural Language Processing (NLP) to homogenize the terms between requirements and models, vocabulary mismatch continues to be an issue that must be taken into account. Since the in-house terms that are used in a specific domain or company are not known synonyms, these in-house terms may not be included in NLP, causing vocabulary mismatch. For example, the terms *PLC* and *system* may be recognized as synonyms, but the terms *PLC* and *COSMOS* are definitely not known to be synonyms because *COSMOS* is an in-house term that is used exclusively by our industrial partner to refer to *PLC*.

As in the tacit knowledge issue, TLR-IR and TLR-Linguistic are seriously affected by vocabulary mismatch because both of them depend, to a large extent, on the text of the requirements. If the terms that are used in the requirements and the terms that are used in the models are not known synonyms, they cannot be related, and therefore the requirement cannot be correctly related to the elements of the model. Therefore, the lack of awareness that is caused by vocabulary mismatch makes it impossible to locate the elements from the model that are relevant to the requirement.

In contrast, TLR-ELtoR is less sensitive to vocabulary mismatch for the same reason described for the tacit knowledge issue. The evaluation of TLR-ELtoR depends on the information provided by training. If the information that is extracted through the training indicates that a term of the requirement is related to a term of an element in the model, the classifier learns that both terms are related to each other even when they are not considered synonyms. Therefore, TLR-ELtoR depends, to a lesser extent, on the synonyms than TLR-IR and TLR-Linguistic, which leads to our approach having better results than the baselines.

### *8.2.3. Available Documentation*

Since TLR-FNN and TLR-RNN are trained using the same knowledge base than TLR-ELtoR, they should also be less sensitive to tacit knowledge and vocabulary mismatch. However, our knowledge base may be unsuitable for properly training a Neural Network. For example, in [9], the training set is composed of 45% of the 769,366 artifacts, so this training set contains about 423,151 feature vectors. However, our training set is composed of the encoding of the knowledge base that has 103 samples whose model fragments have around 15 elements. Therefore, since the ending is performed at the model-element level, the training set contains about 1545 (103 x 15) feature vectors.

Some works analyze the impact of the number of samples on the performance of the neural networks. The authors in [73, 74] suggest the use of a minimum of 10–30p samples for training, where p is the number of features vectors used. However, this rule is often universally enforced in remote sensing without questioning its relevance to the complexity of the specific problem [75]. In fact, in some domains, the best result are obtained with 2p or 4p samples for training [76, 75]. Therefore, a small knowledge base may be insufficient and a large knowledge base may introduce noise.

On the other hand, the knowledge base may also be affected by the vaporization problem [77]. In fact, some industrial companies do not store enough information to create a knowledge base with the necessary completeness and size. However, these domains also need to recover the traceability links, and our approach can be successfully used even if the knowledge base is small, as our evaluation proves.

Since TLR-LtoR is based on LtoR as TLR-ELtoR also is, we might expect that TLR-LtoR will not have the problems described. However, TLR-LtoR obtained the worst results because the search space was too big, so the exploration of this search space randomly required many more iterations. Therefore, by evaluating the same number of model fragments using the two approaches, the TLR-ELtoR obtained the best results thanks to the combination of the LtoR, which provides a successful evaluation of the model fragments, and the evolutionary algorithm, which allows the search space to be explored in an effective way.

## 9. Threats to Validity

In this section, we use the classification of threats to validity of [78] to acknowledge the limitations of our approach.

**Construct validity:** This aspect of validity reflects the extent to which the operational measures that are studied represent what the researchers have in mind. To minimize this risk, our evaluation is performed using four measures: precision, recall, F-measure, and MCC. These measures are widely accepted in the software engineering research community.

**Internal Validity:** This aspect of validity is of concern when causal relations are examined. There is a risk that the factor being investigated may be affected by other neglected factors. RankBoost tends to overfit when the knowledge base is not large enough and there are many encoding features [79]. The number of samples in our knowledge base may look small; however, this threat has been reduced because our approach uses only 54 encoding features, which is a small number in machine learning applications [31, 32].

**External Validity:** This aspect of validity is concerned with to what extent it is possible to generalize the findings, and to what extent the findings are of relevance for other cases. Both requirements and models are frequently leveraged to specify all kinds of different software. The requisites for applying our approach are that the set of models where the requirements must be located conform to MOF (the OMG metalanguage for defining modeling languages), and that the requirements must be provided using natural language. Our experiment does not rely on the specific conditions of our domain. Nevertheless, the experiment and its results should be replicated in other domains before assuring their generalization.

**Reliability:** This aspect is concerned with to what extent the data and the analysis are dependent on the specific researchers. To reduce this threat, the knowledge base, the requirements descriptions, and the product models were provided by our industrial partner.

## 10. Related Work

Works that are related to our research are mainly found within the knowledge area of Traceability Link Recovery. In a more general fashion, works in the knowledge area of Feature Location can be relevant for our research as well. Feature Location takes a query as input and returns its materialization as a result. Potentially, the techniques that are applied to locate features could also be applied to Traceability Link Recovery for requirement queries. In this section, we analyze some of the existing approaches in both areas and compare our work with these approaches.

### *10.1. Traceability Link Recovery*

There are several TLR techniques in use that utilize requirement queries. Most of them deal with source code or focus on the usage and impact of the specific techniques in use. This section analyzes these kinds of works and differentiates our work from them.

Most of the existing works focus on Traceability Link Recovery between requirements and source code. CERBERUS [80] provides a hybrid technique that combines information retrieval, execution tracing, and prune dependency analysis allowing the tracing of requirements to source code. Eaddy et al. [81] present a systematic methodology for identifying which code is related to which requirement, and a suite of metrics for quantifying the amount of crosscutting code. Marcus and Maletic [82] use LSI for recovering the traceability relations between source code and documentation (manuals, design documentation, requirement documents, test suites, etc.). Antoniol et al. [41] propose a method based on information retrieval to recover traceability links between source code and free text documents, such as requirement specifications, design documents, manual pages, system development journals, error logs, and related maintenance reports. Zisman et al. [83] automate the generation of traceability relations between textual requirement artifacts and object models using heuristic rules. These approaches recover the traceability between source code and requirements. In contrast, our work recovers the traceability between requirements and models.

In [22], the authors propose an evolutionary ontological encoding approach to enable Machine Learning techniques to be used to perform Software Engineering tasks in models. Their proof of concept consists of recovering traceability links between requirements and model fragments. However, in the real world whole models are available, rather than model fragments. Therefore, the application of their approach to a real-world problem may be impossible or hard. In contrast, our work recovers the traceability links between requirements and whole models thanks to the evolutionary algorithm that generates and maintains the population of model fragments from whole models.

Some other works focus on the impact and application of Linguistics to TLR problem resolution at several levels of abstraction. Works like [84, 85] or [86], among many others, use Linguistic approaches to tackle specific TLR problems and tasks. In [87], the authors use Linguistic techniques to identify the equivalence between requirements, also defining and using a series of principles for evaluating their performance when identifying equivalent requirements. The authors of [87] conclude that, in their field, the performance of Linguistic techniques is determined by the properties of the given dataset over which they are performed. They measure the properties as a factor to adjust the Linguistic techniques accordingly and then apply their principles to an industrial case study. The work presented in [88] uses Linguistic techniques to study how changes in requirements impact other requirements in the same specification. In their work, the authors analyze TLR between requirements and use Linguistic techniques to determine how changes in requirements must propagate.

Our work differs from [89, 84, 85, 86] since our approach is not based or focused on Linguistic techniques as a means of TLR analysis; instead, we use an evolutionary algorithm to perform TLR between requirements and models, using Linguistic techniques only as a baseline for our work. Moreover, we do not study how Linguistic techniques must be tweaked for specific problems as [87] does. In addition, in contrast to [88], we do not tackle changes in requirements on TLR between requirements, but instead focus our work on TLR between requirements and a set of evolving models.

Some recent works focus on improving TLR results through Neural Networks. Guo et al. [9] present a solution to improve the current automated techniques, which fail to understand the semantics of the software artifacts or to integrate domain knowledge into the tracing process. Therefore, they tend to deliver imprecise and inaccurate results. Specifically, they utilizes Word Embedding and Recurrent Neural Network (RNN) models to generate trace links, which contain the requirements artifact semantics and the domain knowledge. Zhao et al. [90] propose training deep neural networks for generating text-based knowledge in software repositories to improve the accuracy of TLR. The authors in [91] present some challenges in traceability and some of their proposals consider solving these traceability issues through neural networks. In our work, we do not use Neural Networks to improve the results or to perform TLR; instead, we use them as a baseline for comparison with our main line of work, TLR between requirements and models. In addition, we do not address the traceability between source codes and requirements as most of these works do.

Finally, other works target the application of LSI to TLR tasks. De Lucia et al. [11] present a Traceability Link Recovery method and tool based on LSI in the context of an artifact management system, which includes models. The work in [92] takes in consideration the possible configurations of LSI when using the technique for TLR between requirements artifacts, namely requirements and test cases. In their work, the authors state that the configurations of LSI depend on the datasets used, and they expect to be able to determine automatically an appropriate configuration for LSI for any given dataset. In our work, we do not use LSI to perform TLR; instead, we use it as a baseline for comparison with our main line of work, TLR between requirements and models. In addition, we do not tackle different LSI configurations or how LSI configurations impact the results of TLR between requirements and models as [92] does.

### *10.2. Feature Location*

There are several Feature Location techniques and approaches that are applied to locate features for requirement queries, and, as such, can potentially be used for TLR purposes. This section covers these sorts of works, and compares our work with those.

Typechef [93] provides an infrastructure to locate the code that is associated to a given feature by means of analyzing the #ifdef directives. Trace analysis [94] is a run-time technique that is used to locate features. When the technique is executed, it produces traces indicating which parts of code have been executed. Some approaches related to Feature Location use LSI to extract the code that is associated to a feature [95, 96]. These techniques have generally been applied to search the code of a feature in a given individual product. In contrast, our approach searches for model fragments that implement a requirement.

Feature Location approaches in a product family, such as the one presented in [97], center their efforts on finding the code that implements a feature among the different products by combining techniques such as FCA [98] and LSI. In our approach, we are not interested in the code representation of a feature in the family but in locating the most relevant model fragments that implement a requirement. Other works such as [99] focus on applying reverse engineering to the source code to obtain the variability model. In [100], the authors use propositional logic, which describes the dependencies between features. In [101], the authors combine Typechef and propositional logic to extract conditions among a collection of features. These works explicitly engage the variability of products, but they do not indicate the most relevant model fragments for the development of requirements, as our work does.

In [102], Lapeña et al. use Linguistic techniques in combination with an adapted two-step LSI to obtain rankings of methods for all of the requirements of a new product in a product family. The scope of our work is centered around finding model fragments that can be used to implement a specific requirement, while [102] focuses on finding relevant code for the implementation methods of all of the new requirements in a new product in a family.

In addition, even though we had of all the necessary artifacts to apply our approach, the results may not be as good as possible. In fact, some properties of the artifacts have an impact on the results. For example, if there is not enough information in the knowledge base, TLR-ELtoR would not train properly, so the results would be worse than expected. Table 6 shrequirements.

Font et al. [103] use a Single Objective Evolutionary Algorithm (SOEA) to locate features among a family of models in the form of a variation point. Their approach is refined in [15], where the authors use a SOEA to find sets of suitable feature realizations. The authors first cluster model fragments based on their common attributes into feature realization candidates through Formal Concept Analysis, and then LSI ranks the candidates based on the similarity with the feature description. In contrast, our presented approach locates model fragments for requirements instead of variation points for features. In addition, the approaches by Font et al. use FCA and LSI, while our approach trains a LtoR classifier from legacy products to guide the evolutionary algorithm.

Several approaches rely on evolutionary Algorithms guided by LSI for Feature Location. In [57], Font et al. performed a comparison among five different evolutionary algorithms for feature location in models, showing that the best results were achieved by a hybrid between an evolutionary algorithm and a hill climbing technique. In [16], they explored a new direction: taking advantage of already long-living software systems (designed with sustainability in mind) to address the challenge of feature location. Specifically, they used commonality and modifications fitness through model retrospectives in order to promote model fragments that have undergone less modification over time. In constrast, in our approach, the target of the evolutionary algorithm is TLR instead of Feature Location. Moreover, our algorithm is guided by LtoR instead of LSI, which according to our results is less sensitive to the tacit knowledge issues of requirements.

Some approaches rely on LtoR algorithms to locate features in the code [104, 105]. Tien-Duy et al. focus on LtoR using feature vectors that are based on likely invariants. Xin et al. focus on the terms that are defined in a vocabulary to build the feature vectors. In our approach,we use LtoR as an objective for the evolutionary algorithm and perform TLR between requirements and models instead of Feature Location in code.

Other works rely on ontologies to locate features in code. In [106], a systematic approach is used to locate features by using ontology fragments. Hayashi et al. [107] propose an ontology-based technique to locate features that are defined by natural language descriptions. Ratiu et al. [108] present a framework to recover the mappings between entities from an ontology and program elements. Petrenko et al. [109] perform a study about the performance of programmers when they locate features by using ontology fragments. In contrast, our approach performs TLR between requirements and models, using an evolutionary algorithm that is guided by a LtoR classifier.

Finally, the works presented in [110, 111, 112] focus on the location of features in models using comparisons among models in a family of models. Zhang et al. [110] propose a generic approach to locate the feature realizations by exploring the commonality and the variability of models through their automatic comparison. In [111], the approach is refined to reduce the manual effort required in the formalization of the feature realizations when new product models are included in a product line. In the approach presented in [112], the variability between models is determined using an exchangeable metric, taking into account different attributes of the models. However, all of these approaches are based on the location of features through comparisons among the models, while our approach performs TLR between requirements and models. In addition, we do so by relying on an evolutionary algorithm guided by a LtoR classifier.

## 11. Replication of the Results

The implementation for our approach is available at `http://bitbucket.org/svitusj/flame`. We have also made the dataset available in the same url. The dataset contains the requirements and the models that are used in our experiment as well as the knowledge base with the requirements and the model fragments that are used to train the classifier. The implementation for the five baselines is also available at the same location. Therefore, our public online repository contains the source code of our approach, the source code of the two baselines, and the dataset (requirements and models).

## 12. Conclusions

Both Evolutionary Algorithms and Learning to Rank algorithms have a wide range of successful applications, but current research efforts have so far neglected the application of the two on Traceability Link Recovery (TLR) between requirements and models. In this paper, we propose the TLR-ELtoR approach, which recovers traceability links between the requirements of a software system and its models by leveraging the usage of an evolutionary algorithm (EA) that is guided by a Learning to Rank (LtoR) algorithm.

We evaluated our TLR-ELtoR approach in terms of precision, recall, the F-measure, and the Matthews Correlation Coefficient. To do this, we compared it to five baselines in an industrial domain (firmware of train PLCs with CAF). The first baseline is a Linguistic Rule-Based (TLR-Linguistic) approach that is based on Parts-of-Speech (POS) tagging and traceability rules. The second one is an Information Retrieval (TLR-IR) approach that is based on Latent Semantic Indexing (LSI) and Singular Value Decomposition (SVD). The third one is a Feedforward Neural Network (TLR-FNN) approach that is based on a traditional neural network structure. The fourth one is a Recurrent Neural Network (TLR-RNN) approach that is based on an extension of a Feedforward Neural Network with feedbaack connections to model the temporal characteristics of the problem being learned [13]. The fifth one is a Learning to Rank (TLR-LtoR) approach based on ranking Machine Learning algorithms of the same name. We report our evaluation, including: experimental setup, results, statistical analysis, and threats to validity.

The results show that the application of an evolutionary algorithm guided using the LtoR algorithm by means of TLR-ELtoR pays off for TLR. The results also show that our approach can be applied in real-world environments. The statistical analysis of the results assesses the level of the improvement that our approach offers. Moreover, the discussion shows how our approach is limited by the available documentation and how our approach may be beneficial for dealing with issues such as tacit knowledge and vocabulary mismatch.

We acknowledge that we could have proposed other approaches as baselines. For example, instead of using LSI or SVD, we could have used LtoR or other machine learning technique to support TLR-IR in order to check the relevance of a model element with a requirement. The baselines were selected according to their performance, taking into account a classification of approaches for TLR in models, or their recent successful, taking into account recent approaches for TLR. However, a future work could consist of a deeper comparison of our approach with other approaches for TLR, where the contribution could be not only the comparison with other alternative baselines but also the discussion of the advantages and limitations of these alternatives for TLR in models.

*Acknowledgments.*

This work has been developed with the financial support of the Spanish State Research Agency and the Generalitat Valenciana under the projects DataME TIN2016-80811-P, ALPS RTI2018-096411-B-I00, ACIF/2018/171 and PROMETEO/2018/176, and co-financed with ERDF.

## References

[1] O. C. Gotel, C. Finkelstein, An Analysis of the Requirements Traceability Problem, in: Proceedings of the First International Conference on Requirements Engineering, IEEE, 1994, pp. 94–101.

[2] G. Spanoudakis, A. Zisman, Software Traceability: a Roadmap, Handbook of Software Engineering and Knowledge Engineering 3 (2005) 395–428.

[3] R. Watkins, M. Neal, Why and How of Requirements Tracing, IEEE Software 11 (4) (1994) 104–106.

[4] A. Ghazarian, A Research Agenda for Software Reliability, IEEE Reliability Society 2009 Annual Technology Report.

[5] P. Rempel, P. Mäder, Preventing Defects: the Impact of Requirements Traceability Completeness on Software Quality, IEEE Transactions on Software Engineering 43 (8) (2017) 777–797.

[6] R. M. Parizi, S. P. Lee, M. Dabbagh, Achievements and Challenges in State-of-the-Art Software Traceability between Test and Code Artifacts, IEEE Transactions on Reliability 63 (4) (2014) 913–926.

[7] S. Winkler, J. Pilgrim, A Survey of Traceability in Requirements Engineering and Model-Driven Development, Software and Systems Modeling (SoSyM) 9 (4) (2010) 529–565.

[8] A. Rummler, B. Grammel, C. Pohl, Improving Traceability in Model-Driven Development of Business Applications, in: ECMDA Traceability Workshop (ECMDA-TW), 2007, pp. 7–15.

[9] J. Guo, J. Cheng, J. Cleland-Huang, Semantically Enhanced Software Traceability using Deep Learning Techniques, in: Software Engineering (ICSE), 2017 IEEE/ACM 39th International Conference on, IEEE, 2017, pp. 3–14.

[10] G. Spanoudakis, A. Zisman, E. Pérez-Minana, P. Krause, Rule-Based Generation of Requirements Traceability Relations, Journal of Systems and Software 72 (2) (2004) 105–127.

[11] A. De Lucia, F. Fasano, R. Oliveto, G. Tortora, Enhancing an Artefact Management System with Traceability Recovery Features, in: Proceedings of the 20th IEEE International Conference on Software Maintenance, IEEE, 2004, pp. 306–315.

[12] A. D. Lucia, F. Fasano, R. Oliveto, G. Tortora, Recovering Traceability Links in Software Artifact Management Systems using Information Retrieval Methods, ACM Transactions on Software Engineering and Methodology (TOSEM) 16 (4) (2007) 13.

[13] A. P. Engelbrecht, Computational Intelligence: An Introduction, 2nd Edition, Wiley Publishing, 2007.

[14] Ø. Haugen, B. Møller-Pedersen, J. Oldevik, G. K. Olsen, A. Svendsen, Adding Standardized Variability to Domain Specific Languages, in: Proceedings of the 12th International Software Product Lines Conference, 2008, pp. 139–148.

[15] J. Font, L. Arcega, Ø. Haugen, C. Cetina, Feature Location in Models Through a Genetic Algorithm Driven by Information Retrieval Techniques, in: Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems, MODELS '16, ACM, 2016, pp. 272–282.

[16] C. Cetina, J. Font, L. Arcega, F. Pérez, Improving Feature Location in Long-Living Model-Based Product Families Designed with Sustainability Goals, Journal of Systems and Software 134 (2017) 261–278.

[17] M. Affenzeller, S. M. Winkler, S. Wagner, A. Beham, Genetic Algorithms and Genetic Programming - Modern Concepts and Practical Applications, CRC Press, 2009.

[18] L. Davis, Handbook of Genetic Algorithms.

[19] Z.-J. Lu, Q. Xiang, Y.-m. Wu, J. Gu, Application of Support Vector Machine and Genetic Algorithm Optimization for Quality Prediction within Complex Industrial Process, in: Industrial Informatics (INDIN), 2015 IEEE 13th International Conference on, IEEE, 2015, pp. 98–103.

[20] M. Bianchini, M. Maggini, L. C. Jain, Handbook on neural information processing, Springer, 2013.

[21] G. Chandrashekar, F. Sahin, A Survey on Feature Selection Methods, Computers & Electrical Engineering 40 (1) (2014) 16–28.

[22] A. C. Marcén, F. Pérez, C. Cetina, Ontological Evolutionary Encoding to Bridge Machine Learning and Conceptual Models: Approach and Industrial Evaluation, in: International Conference on Conceptual Modeling, Springer, 2017, pp. 491–505.

[23] R. Lapeña, J. Font, Ó. Pastor, C. Cetina, Analyzing the Impact of Natural Language Processing over Feature Location in Models, ACM SIGPLAN Notices 52 (12) (2017) 63–76.

[24] A. Shabtai, R. Moskovitch, Y. Elovici, C. Glezer, Detection of Malicious Code by Applying Machine Learning Classifiers on Static Features: A State-of-the-Art Survey, information security technical report 14 (1) (2009) 16–29.

[25] P. Refaeilzadeh, L. Tang, H. Liu, Cross-Validation, in: Encyclopedia of database systems, Springer, 2009, pp. 532–538.

[26] Q. Song, Z. Jia, M. Shepperd, S. Ying, J. Liu, A General Software Defect-Proneness Prediction Framework, IEEE Transactions on Software Engineering 37 (3) (2011) 356–370.

[27] A. S. Sayyad, J. Ingram, T. Menzies, H. Ammar, Scalable Product Line Configuration: A Straw to Break the Camel's Back, in: IEEE/ACM 28th International Conference on Automated Software Engineering (ASE), 2013, pp. 465–474. `doi:10.1109/ASE.2013.6693104`.

[28] Y. Freund, R. Iyer, R. E. Schapire, Y. Singer, An Efficient Boosting Algorithm for Combining Preferences, Journal of machine learning research 4 (Nov) (2003) 933–969.

[29] S. D. Canuto, F. M. Belém, J. M. Almeida, M. A. Gonçalves, A Comparative Study of Learning-to-Rank Techniques for Tag Recommendation, Journal of Information and Data Management 4 (3) (2013) 453.

[30] Z. Cao, Y. Tian, T.-D. B. Le, D. Lo, Rule-Based Specification Mining Leveraging Learning to Rank, Automated Software Engineering (2018) 1–30.

[31] Z.-H. Zhou, J. Feng, Deep Forest: Towards an Alternative to Deep Neural Networks, arXiv preprint arXiv:1702.08835.

[32] J. Wang, P. Zhao, S. C. Hoi, R. Jin, Online Feature Selection and its Applications, IEEE Transactions on Knowledge and Data Engineering 26 (3) (2014) 698–710.

[33] M. F. Kıraç, B. Aktemur, H. Sözer, VISOR: A Fast Image Processing Pipeline with Scaling and Translation Invariance for Test Oracle Automation of Visual Output Systems, Journal of Systems and Software 136 (2018) 266–277.

[34] A. H. Hirzel, G. Le Lay, V. Helfer, C. Randin, A. Guisan, Evaluating the Ability of Habitat Suitability Models to Predict Species Presences, ecological modelling 199 (2) (2006) 142–152.

[35] G. Leech, R. Garside, M. Bryant, CLAWS4: the Tagging of the British National Corpus, in: Proceedings of the 15th Conference on Computational Linguistics - Volume 1, Association for Computational Linguistics, 1994, pp. 622–628.

[36] W. B. Frakes, R. Baeza-Yates, Information Retrieval: Data Structures and Algorithms.

[37] C. D. Manning, P. Raghavan, H. Schütze, et al., Introduction to Information Retrieval, Vol. 1, Cambridge University Press, 2008.

[38] G. Salton, M. J. McGill, Introduction to Modern Information Retrieval.

[39] D. Lucia, et al., Information Retrieval Models for Recovering Traceability Links between Code and Documentation, in: Proceedings of the International Conference on Software Maintenance, IEEE, 2000, pp. 40–49.

[40] R. Oliveto, M. Gethers, D. Poshyvanyk, A. De Lucia, On the Equivalence of Information Retrieval Methods for Automated Traceability Link Recovery, in: 18th International Conference on Program Comprehension, IEEE, 2010, pp. 68–71.

[41] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, E. Merlo, Recovering Traceability Links between Code and Documentation, IEEE Transactions on Software Engineering 28 (10) (2002) 970–983.

[42] A. Marcus, J. I. Maletic, Recovering Documentation-to-Source-Code Traceability Links using Latent Semantic Indexing, in: Proceedings of the 25th International Conference on Software Engineering, IEEE, 2003, pp. 125–135.

[43] A. De Lucia, F. Fasano, R. Oliveto, G. Tortora, Can Information Retrieval Techniques effectively support Traceability Link Recovery?, in: 14th IEEE International Conference on Program Comprehension, IEEE, 2006, pp. 307–316.

[44] T. K. Landauer, P. W. Foltz, D. Laham, An Introduction to Latent Semantic Analysis, Discourse Processes 25 (2-3) (1998) 259–284.

[45] F. Meziane, N. Athanasakis, S. Ananiadou, Generating Natural Language Specifications from UML Class Diagrams, Requirements Engineering 13 (1) (2008) 1–18.

[46] A. Marcus, A. Sergeyev, V. Rajlich, J. Maletic, An Information Retrieval Approach to Concept Location in Source Code, in: Proceedings of the 11th Working Conference on Reverse Engineering, 2004, pp. 214–223. doi:10.1109/WCRE.2004.10.

[47] H. E. Salman, A. Seriai, C. Dony, Feature Location in a Collection of Product Variants: Combining Information Retrieval and Hierarchical Clustering, in: The 26th International Conference on Software Engineering and Knowledge Engineering, 2014, pp. 426–430.

[48] S. Haykin, Neural Networks: a Comprehensive Foundation, Prentice Hall PTR, 1994.

[49] K. Hornik, M. Stinchcombe, H. White, Multilayer Feedforward Networks are Universal Approximators, Neural networks 2 (5) (1989) 359–366.

[50] G. Klambauer, T. Unterthiner, A. Mayr, S. Hochreiter, Self-normalizing Neural Networks, in: Advances in neural information processing systems, 2017, pp. 971–980.

[51] Z. Zhang, L. Chen, P. Tian, J. Su, Source localization in an ocean waveguide using supervised machine learning, Computing 11 5.

[52] J. Mao, W. Xu, Y. Yang, J. Wang, Z. Huang, A. Yuille, Deep Captioning with Multimodal Recurrent Neural Networks (M-RNN), arXiv preprint arXiv:1412.6632.

[53] C. Chopra, S. Sinha, S. Jaroli, A. Shukla, S. Maheshwari, Recurrent Neural Networks with Non-Sequential Data to Predict Hospital Readmission of Diabetic Patients, in: Proceedings of the 2017 International Conference on Computational Biology and Bioinformatics, ACM, 2017, pp. 18–23.

[54] Y. Bengio, P. Simard, P. Frasconi, Learning Long-Term Dependencies with Gradient Descent is Difficult, IEEE transactions on neural networks 5 (2) (1994) 157–166.

[55] S. Hochreiter, J. Schmidhuber, Long Short-Term Memory, Neural computation 9 (8) (1997) 1735–1780.

[56] K. He, X. Zhang, S. Ren, J. Sun, Delving Deep into Rectifiers: Surpassing Human-level Performance on Imagenet Classification, in: Proceedings of the IEEE international conference on computer vision, 2015, pp. 1026–1034.

[57] J. Font, L. Arcega, Ø. Haugen, C. Cetina, Achieving Feature Location in Families of Models through the use of Search-Based Software Engineering, IEEE Transactions on Evolutionary Computation.

[58] C. Beleites, U. Neugebauer, T. Bocklitz, C. Krafft, J. Popp, Sample Size Planning for Classification Models, Analytica Chimica Acta 760 (2013) 25–33.

[59] A. Arcuri, G. Fraser, Parameter Tuning or Default Values? An Empirical Investigation in Search-Based Software Engineering, Empirical Software Engineering 18 (3) (2013) 594–623. `doi:10.1007/s10664-013-9249-9`.

[60] The Stanford Natural Language Processing Group, `https://nlp.stanford.edu/software/tagger.shtml`, [Online; accessed 18-May-2017] (2017).

[61] P. Abeles, Efficient Java Matrix Library, `http://ejml.org/`, [Online; accessed 12-April-2017] (2017).

[62] D. Team, et al., Deeplearning4j: Open-source Distributed Deep Learning for the JVM, Apache Software Foundation License 2.

[63] D. Dyer, The Watchmaker Framework for Evolutionary Computation (Evolutionary/Genetic Algorithms for Java), `http://watchmaker.uncommons.org/`, [Online; accessed 7-April-2016] (2016).

[64] V. Dang, The Lemur Project - Wiki - RankLib, `http://sourceforge.net/p/lemur/wiki/RankLib/`, [Online; accessed April-2017] (2013).

[65] A. Arcuri, L. Briand, A Hitchhiker's Guide to Statistical Tests for Assessing Randomized Algorithms in Software Engineering, Software Testing, Verification and Reliability 24 (3) (2014) 219–250.

[66] S. García, A. Fernández, J. Luengo, F. Herrera, Advanced Nonparametric Tests for Multiple Comparisons in the Design of Experiments in Computational Intelligence and Data Mining: Experimental Analysis of Power, Information Sciences 180 (10) (2010) 2044–2064.

[67] W. Conover, Practical Nonparametric Statistics, 3rd edn Wiley, New York (1999) 250–257.

[68] A. Vargha, H. D. Delaney, A Critique and Improvement of the CL Common Language Effect Size Statistics of McGraw and Wong, Journal of Educational and Behavioral Statistics 25 (2) (2000) 101–132.

[69] T. Joachims, Svmlight: Support vector machine, SVM-Light Support Vector Machine http://svmlight. joachims. org/, University of Dortmund 19 (4).

[70] H. Störrle, On the Impact of Layout Quality to Understanding UML Diagrams: Size Matters, in: International Conference on Model Driven Engineering Languages and Systems, Springer, 2014, pp. 518–534.

[71] I. Rus, M. Lindvall, Knowledge Management in Software Engineering, IEEE Software 19 (3) (2002) 26.

[72] A. Stone, P. Sawyer, Using Pre-Requirements Tracing to Investigate Requirements based on Tacit Knowledge, in: ICSOFT (1), 2006, pp. 139–144.

[73] P. M. Mather, M. Koch, Computer Processing of Remotely-Sensed Images: an Introduction, John Wiley & Sons, 2011.

[74] J. Piper, Variability and Bias in Experimentally Measured Classifier Error Rates, Pattern Recognition Letters 13 (10) (1992) 685–692.

[75] T. G. Van Niel, T. R. McVicar, B. Datt, On the Relationship between Training Sample Size and Data Dimensionality: Monte Carlo Analysis of Broadband Multi-Temporal Classification, Remote Sensing of Environment 98 (4) (2005) 468–480.

[76] S. Walczak, N. Cerpa, Heuristic Principles for the Design of Artificial Neural Networks, Information and software technology 41 (2) (1999) 107–117.

[77] J. S. van der Ven, A. G. Jansen, J. A. Nijhuis, J. Bosch, Design Decisions: The Bridge between Rationale and Architecture, in: Rationale management in software engineering, Springer, 2006, pp. 329–348.

[78] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, A. Wesslén, Experimentation in Software Engineering, Springer Science & Business Media, 2012.

[79] L. Wolf, I. Martin, Robust Boosting for Learning from Few Examples, in: Computer Vision and Pattern Recognition, Vol. 1, IEEE, 2005, pp. 359–364.

[80] M. Eaddy, A. V. Aho, G. Antoniol, Y.-G. Guéhéneuc, Cerberus: Tracing Requirements to Source Code Using Information Retrieval, Dynamic Analysis, and Program Analysis, in: ICPC 2008 conference, IEEE, 2008, pp. 53–62.

[81] M. Eaddy, A. Aho, G. C. Murphy, Identifying, Assigning, and Quantifying Crosscutting Concerns, in: Proceedings of the First International Workshop on Assessment of Contemporary Modularization Techniques, 2007, p. 2.

[82] A. Marcus, J. I. Maletic, Recovering Documentation-to-Source-Code Traceability Links Using Latent Semantic Indexing, in: Proceedings of the 25th International Conference on Software Engineering, IEEE, 2003, pp. 125–135.

[83] A. Zisman, G. Spanoudakis, E. Pérez-Miñana, P. Krause, Tracing Software Requirements Artifacts, in: Software Engineering Research and Practice, 2003, pp. 448–455.

[84] H. Sultanov, J. H. Hayes, Application of Swarm Techniques to Requirements Engineering: Requirements Tracing, in: 18th IEEE International Requirements Engineering Conference, 2010.

[85] S. K. Sundaram, J. H. Hayes, A. Dekhtyar, E. A. Holbrook, Assessing Traceability of Software Engineering Artifacts, Requirements Engineering 15 (3).

[86] C. Duan, J. Cleland-Huang, Clustering Support for Automated Tracing, in: Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering, 2007.

[87] D. Falessi, G. Cantone, G. Canfora, Empirical Principles and an Industrial Case Study in Retrieving Equivalent Requirements via Natural Language Processing Techniques, Transactions on Software Engineering 39 (1).

[88] C. Arora, M. Sabetzadeh, A. Goknil, L. C. Briand, F. Zimmer, Change Impact Analysis for Natural Language Requirements: An NLP Approach, in: IEEE 23rd International Requirements Engineering Conference, 2015.

[89] K. Ryan, The Role of Natural Language in Requirements Engineering, in: Proceedings of IEEE International Symposium on Requirements Engineering, 1993.

[90] Y. Zhao, T. S. Zaman, T. Yu, J. H. Hayes, Using Deep Learning to Improve the Accuracy of Requirements to Code Traceability, Grand Challenges of Traceability: The Next Ten Years (2017) 22.

[91] G. Antoniol, J. Cleland-Huang, J. H. Hayes, M. Vierhauser, Grand Challenges of Traceability: The Next Ten Years, arXiv preprint arXiv:1710.03129.

[92] S. Eder, H. Femmer, B. Hauptmann, M. Junker, Configuring Latent Semantic Indexing for Requirements Tracing, in: Proceedings of the

2nd International Workshop on Requirements Engineering and Testing, 2015.

[93] C. Kästner, P. G. Giarrusso, T. Rendel, S. Erdweg, K. Ostermann, T. Berger, Variability-Aware Parsing in the Presence of Lexical Macros and Conditional Compilation, in: Proceedings of the 26th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, 2011.

[94] A. D. Eisenberg, K. D. Volder, Dynamic Feature Traces: Finding Features in Unfamiliar Code, in: 21st IEEE International Conference on Software Maintenance, 2005.

[95] D. Poshyvanyk, Y. Guéhéneuc, A. Marcus, G. Antoniol, V. Rajlich, Feature Location Using Probabilistic Ranking of Methods Based on Execution Scenarios and Information Retrieval, IEEE Trans. Software Eng. 33 (6).

[96] D. Liu, A. Marcus, D. Poshyvanyk, V. Rajlich, Feature Location via Information Retrieval Based Filtering of a Single Scenario Execution Trace, in: Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering, ACM, 2007, pp. 234–243.

[97] Y. Xue, Z. Xing, S. Jarzabek, Feature Location in a Collection of Product Variants, in: 19th Working Conference on Reverse Engineering, 2012.

[98] B. Ganter, R. Wille, Formal Concept Analysis: Mathematical Foundations, 1st Edition, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1997.

[99] S. She, R. Lotufo, T. Berger, A. Wasowski, K. Czarnecki, Reverse Engineering Feature Models, in: Proceedings of the 33rd International Conference on Software Engineering, 2011.

[100] K. Czarnecki, A. Wasowski, Feature Diagrams and Logics: There and Back Again, in: Proceedings of the 11th International Software Product Lines Conference, 2007.

[101] S. Nadi, T. Berger, C. Kästner, K. Czarnecki, Mining Configuration Constraints: Static Analyses and Empirical Results, in: 36th International Conference on Software Engineering, 2014.

[102] R. Lapeña, M. Ballarín, C. Cetina, Towards Clone-and-Own Support: Locating Relevant Methods in Legacy Products, in: Proceedings of the 20th International Conference on Software Product Lines, 2016.

[103] J. Font, L. Arcega, Ø. Haugen, C. Cetina, Feature Location in Model-Based Software Product Lines Through a Genetic Algorithm, in: Proceedings of the 15th International Conference on Software Reuse: Bridging with Social-Awareness, 2016, pp. 39–54.

[104] X. Ye, R. Bunescu, C. Liu, Learning to Rank Relevant Files for Bug Reports using Domain Knowledge, in: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, ACM, 2014, pp. 689–699.

[105] T.-D. B. Le, D. Lo, C. Le Goues, L. Grunske, A Learning-to-Rank Based Fault Localization Approach using Likely Invariants, in: Proceedings of the 25th International Symposium on Software Testing and Analysis, ACM, 2016, pp. 177–188.

[106] L. A. Wilson, Using Ontology Fragments in Concept Location, in: IEEE International Conference on Software Maintenance, 2010, pp. 1–2.

[107] S. Hayashi, T. Yoshikawa, M. Saeki, Sentence-to-Code Traceability Recovery with Domain Ontologies, in: 2010 Asia Pacific Software Engineering Conference, 2010, pp. 385–394.

[108] D. Ratiu, F. Deissenboeck, From Reality to Programs and (not quite) Back Again, in: 15th IEEE International Conference on Program Comprehension, IEEE, 2007, pp. 91–102.

[109] M. Petrenko, V. Rajlich, R. Vanciu, Partial Domain Comprehension in Software Evolution and Maintenance, in: 16th IEEE International Conference on Program Comprehension, 2008, pp. 13–22.

[110] X. Zhang, Ø. Haugen, B. Moller-Pedersen, Model Comparison to Synthesize a Model-Driven Software Product Line, in: 15th International Software Product Line Conference (SPLC), IEEE, 2011, pp. 90–99.

[111] X. Zhang, Ø. Haugen, B. Møller-Pedersen, Augmenting Product Lines, in: 19th Asia-Pacific Software Engineering Conference, 2012.

[112] D. Wille, S. Holthusen, S. Schulze, I. Schaefer, Interface Variability in Family Model Mining, in: 17th International Software Product Line Conference, 2013.