Learning to Classify Non-Functional Requirements: A Systematic Review

Paulo Vítor C. F. Libório[a,*], Kevin Abreu de Araujo[a], Anderson Oliveira[a], Juliana Alves Pereira[a], Alessandro Fabricio Garcia[a]

[a]*Departament of Informatics, Pontifical Catholic University of Rio de Janeiro (PUC-Rio), Rio de Janeiro, Brazil*

Abstract

Non-functional requirements (NFRs) are used to explain how the system should perform. They are essential for the success of a project. Failure to meet NFRs can result in user frustration and potential abandonment of the system (*e.g.* if a system is too slow or not secure). Recent advancements in learning approaches have enabled the identification and classification of NFRs, thus providing insights into developers' concerns during system development, maintenance, and evolution. However, a comprehensive catalog and comparison of the latest open-source approaches are currently missing. This study presents a systematic literature review of 22 approaches. Our objective is to understand the learning approaches used, their strengths and weaknesses, and how they are evaluated. Based on our review, we created an open-source, robust, and unified dataset of manually labeled NFRs and learning approaches.

Keywords: systematic literature review, requirement engineering, non-functional requirements, machine learning, natural language processing

*Corresponding author.

Email addresses: pliborio@inf.puc-rio.br (Paulo Vítor C. F. Libório), karaujo@icad.puc-rio.br (Kevin Abreu de Araujo), anderson.jose.so@inf.puc-rio.br (Anderson Oliveira), juliana@inf.puc-rio.br (Juliana Alves Pereira), afgarcia@inf.puc-rio.br (Alessandro Fabricio Garcia)

1. Introduction

Non-functional requirements (NFRs) are system characteristics that specify how well a software system meets non-functional objectives, as opposed to what functions it performs. *Usability, Security, Performance, Reliability, Scalability*, and *Maintainability* are examples of NFRs. NFRs are critical for software systems to ensure customer satisfaction, market competitiveness, and regulatory compliance. However, software developers and stakeholders often neglect, misunderstand, or misspecificate NFRs [1]. This can lead to various problems, such as low quality, high cost, delayed delivery, increased risk, and reduced value of software systems. [2] Therefore, effective techniques and tools are needed to obtain, analyze, specify, verify, and manage NFRs throughout the software development life cycle [3].

One of the key challenges in dealing with NFRs is their diversity and complexity. NFRs can vary in their definition, scope, level of abstraction, granularity, priority, and interdependence [2]. NFRs can also conflict or trade-off with each other or with functional requirements [4]. These characteristics make identifying, understanding, and communicating NFRs among different stakeholders and software artifacts difficult. To address this challenge, NFR classification is proposed to identify, organize, and label NFRs into meaningful categories or types [3]. By providing a common vocabulary, a structured framework, and consistent notation for NFRs, it can help improve understanding, communication, and management of NFRs [5]. NFR classification can also help with NFR analysis and evaluation by allowing measurement and aggregation of NFRs in multiple dimensions.

To support NFR classification, several state-of-the-art machine learning solutions, and tools have been proposed to automatically classify NFRs and assist software developers [6, 7, 8, 9, 10, 11, 12, 13, 14, 15]. These solutions and tools use various algorithms and techniques to extract features from natural language requirements and assign them to predefined NFR types. However, these solutions and tools have some limitations and challenges: (1) they may suffer from low accuracy, scalability, usability, or applicability due to the diversity and complexity of NFRs; (2) most of the solutions are not transferred among different types of NFRs; (3) the lack of NFR definition standardization make it hard to transfer the knowledge even for the same type of NFR; (4) for each different context and NFR, it requires an exhaustive manual labeling process; (5) the quality of the available datasets that are used to train, test, and validate the state-of-the-art solutions are also problematic.

In a previous review, Binkhonain and Zhao [16] evaluated 24 studies about classification of NFRs, with the latest study being from 2017. Since then, new techniques have been presented in the literature. The goal of this work is to provide a comprehensive overview of current approaches for NFR classification in software development, by conducting a systematic review of 22 recent studies in the field. The results of this review provide valuable insights into the current trends, gaps, and opportunities for future research and practice in NFR classification.

We make the following five contributions:

1. We identified 19 main different NFRs: *Usability, Security, Performance, Maintainability, Availability, Scalability, Portability, Look and Feel, Fault Tolerance, Legal, Operational, Operability, Reliability, Compatibility, Safety, Verifiability, Functional Suitability, Capability* and *Efficiency*.

2. We described 16 learning techniques and 5 evaluation metrics used in the literature.

3. As case studies, we identified a portfolio of 11 real-world datasets that contain several systems requirement specifications targeting several domains and NFRs. We relate and discuss the learning and evaluation results of state-of-the-art solutions across these datasets.

4. We discussed the results obtained in our literature review against those obtained by Binkhonain and Zhao [16]. Moreover, we identified a set of open challenges faced by the current solutions, in order to guide researchers and practitioners to experiment with state-of-the-art solutions and build new solutions.

5. We build a Web repository[1] to make our SLR results publicly available for the purpose of reproducibility and extension.

The rest of the study is structured as follows: Section 2 summarizes the background and related work. Section 3 describes the methodology used in our literature review. Section 4 discusses the research questions and our results. Section 6 discusses the threats to the validity of our study. Section 7 concludes our study, presenting our future directions.

[1] `https://github.com/pvliborio/slr-supplementary-material`

2. Background and Related Work

In this section, we introduce basic concepts used throughout the study, as well as related works in the field.

2.1. Functional and Non-Functional Requirements

Every software starts with the specification of their requirements. IEEE Standard 610.12-1990 [17] defines requirements as a condition or capability of a system. They are necessary attributes of a system and can define characteristics or the quality of the system [1]. We can divide the requirements of a system into two types: *functional requirements* (FRs) and *non-functional requirements* (NFRs).

Generally, FRs are defined as *what* the system is expected to perform [4] or a function that the system has to execute [17]. They define the desired behavior of the system. For example, *"The system shall authorize a user if their name and password match those of the database"*. This requirement defines the basic authorization function of a system.

NFRs are defined as *how* the system is expected to perform. However, the definition of NFRs is not a consensus. Some authors define NFRs as attributes or constraints to a system [2, 4]. Others define NFRs as properties of a system [18]. For example, an NFR related to security can be defined as *"The password must be hashed using Argon2id"*. Nevertheless, it is generally accepted that failure to meet NFRs may become a threat to a software [19]. In this example, if the software did not use any password hash function or an insecure one, it may expose the user account to the threat of unauthorized access.

2.2. Machine Learning

The goal of achieving the same level of intelligence as a human being has been a common problem in the Computer Science area, this field of study is called Artificial Intelligence (AI). A subfield of AI is called Machine Learning (ML) [20].

ML algorithms can develop models that learn from historical data to make predictions. The process generally involves the following steps: *(i)* sampling: the collection of the data for learning and testing; *(ii)* labeling: the annotation of the sampled data; *(iii)* learning: the use of learning algorithms to train a model to make predictions; and *(iv)* evaluation: the use of the testing data to evaluate the model. Following, we describe each of these steps.

2.2.1. Sampling

An ML algorithm can only learn from data. In this step, sufficient data is collected for the algorithm to learn a pattern. For example, in the field of requirements engineering, it can be done by scrapping a site, extracting it from a *Software Requirement Specification* (SRS) document, using an API, or getting it from an already created dataset [21, 22]. However, not always can the entire data of a dataset be usable. If the dataset is large enough, manually labeling it is not viable. Thus, sampling techniques, *e.g.* heuristics, are used to select a subset of data. This can be achieved by selecting SRS using keywords or by calculating the similarity between items from the dataset.

The sampling data will be split into two sets: a set to train and a set to test a model. This is needed because after the model learns how to classify, it needs to be tested on already classified data for evaluation.

2.2.2. Labeling

After sampling the data, it is necessary to label the data. The label is used for supervised learning algorithms (see Section 2.2.3). For example, an expert can label a sentence as being related to the NFR performance or maintainability. Labeling data is generally a manual task conducted by experts [22]. As mentioned in Section 4.2, most of the datasets used in the literature use all available instances in an SRS, without using sampling.

2.2.3. Learning

The next step is to decide which algorithm is going to be used. In the following, we introduce some concepts and review some algorithms used in the literature.

Types of Learning for ML Solutions:

- **Supervised Learning (SL)**: uses data with labels to teach algorithms how to classify or predict things [23]. It can be for categories or numbers. Some examples are linear regression, logistic regression, decision trees, and support vector machines (SVMs).

- **Unsupervised Learning (USL)**: uses data without labels to teach algorithms how to find patterns or structures in the data [23]. It can be used for grouping, finding rules, or reducing features. Some examples are principal component analysis (PCA), autoencoders, market basket analysis, and k-means clustering.

- **Semi-supervised Learning (SSL)**: it is a hybrid approach that mixes supervised and unsupervised learning. It uses data with and without labels to teach algorithms how to classify or predict things. Some examples are label propagation, label spreading, self-training, and co-training.

We will now present some of the algorithms used in the literature.

— *Naive Bayes (NB).* It is an ML algorithm that uses probability theory to classify data into different categories. It is based on Bayes' theorem, which calculates the likelihood of a category given some features or evidence [24]. Naive Bayes is called 'naive' because it assumes that features are independent of each other, which may not be true in reality. However, this simplifies the computation and makes the algorithm fast and easy to implement.

Throughout our review, the specified implementations of this algorithm were:

- **Multinomial**: A multinomial Naive Bayes implementation assumes a multinomial distribution for all its features [24], suitable for classification with discrete features such as word counts for text classification.
- **Gaussian**: A Gaussian Naive Bayes implementation assumes a Gaussian (normal) distribution for all its features [24], suitable for classification with continuous features such as the frequency of a word or phrase in a document.
- **Bernoulli**: A Bernoulli Naive Bayes implementation assumes a Bernoulli (binary) distribution for all its features [24], suitable for classification with binary features such as the presence or absence of a word in a document.

— *SVM.* Stands for Support Vector Machine, is an ML algorithm that uses geometry to separate data into categories. It finds the best line or plane with the largest gap between the nearest points of different categories [22]. SVM can handle multiclass classification using methods such as the One vs. One scheme used by scikit-learn[2]. This approach works by using a binary classifier for each pair of classes, combined with a voting scheme, to decide the final result of the classification. SVM is used mainly for tasks such as image recognition, face detection, and handwriting recognition.

[2]https://scikit-learn.org/stable/

— *KNN.* Stands for K-Nearest Neighbours, and is an ML algorithm that uses the proximity of data points to classify them into different categories. It works by finding the k points closest to a new point and assigning it to the most common category among them. KNN can handle both classification and regression problems, but is more commonly used for classification [24].

— *CNN.* Stands for Convolutional Neural Network, which is a type of ML algorithm that uses multiple layers of artificial neurons to learn characteristics and patterns from data [25]. For NFR classification, it applies convolutional filters, which are special functions that slide over sentences and detect patterns, to sentences, which extract features that may capture the meaning and sentiment of words, regardless of their order, form, or length.

— *BERT-CNN.* It's a novel neural network model that combines BERT and CNN to process natural language input. It uses BERT to create word vectors that capture the meaning of the words and the input sequence. Then it uses CNN to extract local features and reduce the size of the data. The model produces a vector representation of the input and a probability distribution over the classes. The model takes advantage of global and local input characteristics and improves the performance of natural language processing tasks, such as NFR classification [26].

— *Logistic Regression (LR).* It is a type of ML algorithm that uses linear and logistic functions to classify data into different categories. It is based on estimating the probability of a category given some features or evidence and comparing it with a threshold value [22]. LR is widely used for applications such as medical diagnosis, credit scoring, and marketing analysis.

— *Tree-based Algorithms (TbA).* Tree-based algorithms employ trees to gain knowledge from data. They divide the data into smaller and more similar clusters according to certain criteria. One criterion is information gain, which evaluates how much we can learn about the result by dividing it according to an attribute. Another criterion is the Gini index, which assesses how unequal or impure a group is. The most effective splits are those with high information gain and a low Gini index[24].

Throughout our review, the specified implementations of this algorithm were:

- Decision Trees: They are based on creating a single tree that splits the data at each node until a leaf node is reached [22]. Decision trees can be pruned to reduce complexity and improve generalization.

- Random Forests: They are based on creating multiple trees from different bootstrap samples, which are smaller samples drawn randomly with data replacement, randomly selecting a subset of characteristics in each split [24]. Random Forests can reduce the variance and bias of single trees, and increase their accuracy and robustness.

— *LSTM/Bi-LSTM.* Stands for Long Short-Term Memory, which is a type of recurrent neural network (RNN) that can learn long-term dependencies from sequential data, such as text, speech, or video. It has "memory cells" that store and forget values and gates that control the information flow. There are three gates: input, forget, and output. The input gate adds new values to the memory cell. The forget gate erases old values from the memory cell. The output gate outputs information from the memory cell [25].

Alternatively, Bi-LSTM stands for Bidirectional Long-Short-Term Memory, which is a variation of LSTM that can learn from both past and future contexts of sequential data. It is based on the use of two LSTMs that process the data in opposite directions and concatenate or add their outputs[27]. Bi-LSTM can capture more data and enhance accuracy; however, it may be more difficult to parallelize due to the fact that it requires both forward and backward passes of the input sequence, which are interdependent. This implies that the computation of one direction cannot begin until the other direction is completed, or at least until some intermediate results are accessible. This creates a sequential bottleneck that restricts the parallelization potential of Bi-LSTM.

— *Stochastic Gradient Descent (SGD).* It is a type of optimization technique used to train ML models. It is based on updating the model parameters using the gradient of the objective function calculated from a small subset of the data, instead of using the entire data set [25].

— *Subspace Discriminant (SD).* It is a type of ML algorithm that works by first finding the subspaces that are most discriminative between the different classes[28]. This is done by maximizing the ratio of the scatter matrix between classes to the scatter matrix within classes.

— *Artificial Neural Network (ANN).* It is a type of ML algorithm that uses multiple layers of artificial neurons to learn features and patterns from data [25]. It is inspired by the structure and function of the biological neural network in the brain. ANN can handle complex and high-dimensional data, such as images, videos, and natural language.

— *TPOT.* Standing for Tree-Based Pipeline Optimization Tool, TPOT is a Python library for automated ML. It uses a tree-based structure to represent a model pipeline for a predictive modeling problem, including data preparation and modeling algorithms and model hyperparameters [29]. TPOT uses a genetic programming approach to maximize the pipeline and find the most suitable one for your data. Constructed on top of scikit-learn, TPOT is compatible with a variety of ML models and techniques for feature extraction, preprocessing, and parameter optimization for your dataset. For example, a normal TPOT return can be as follows:

> "Best pipeline:
> DecisionTreeClassifier(RBFSampler(input_matrix, RBFSampler__gamma=0.85),
> DecisionTreeClassifier__criterion=entropy, DecisionTreeClassifier__max_depth=3,
> DecisionTreeClassifier__min_samples_leaf=4,
> DecisionTreeClassifier__min_samples_split=9)"
>
> (The TPOT documentation has a page of Jupyter Notebook examples[a])
>
> [a] `http://epistasislab.github.io/tpot/examples/`

It's also possible to export the optimized solution into a defined path, and TPOT will generate the Python file for the entire pipeline of your solution. Furthermore, TPOT is an open source project that is still in the process of being developed[3].

— *Zero-Shot Learning (ZSL).* It is a learning paradigm that aims to perform tasks without using any labelled training data. [30] ZSL can be applied to various natural language processing tasks, such as text classification, entity recognition, and relation extraction. The main idea of ZSL is to leverage pretrained language models that are trained on large amounts of unlabeled text data, such as Wikipedia or app reviews, and use them to generate representations for both the input text and the class labels. Then, ZSL compares the

[3] `https://github.com/EpistasisLab/tpot`

similarity between the text and the labels and assigns the text to the most similar label. ZSL can handle both seen and unseen classes, which means that it can classify text into classes that are not labeled in training data.ZSL has several advantages over supervised learning, such as reducing the cost and effort of data annotation and allowing flexibility and generalization to different tasks and domains.

— *Label Propagation (LP).* It is a semi-supervised ML algorithm that assigns labels to previously unlabeled data points. It is based on creating a graph that connects the data points based on their similarity and propagating the labels from the labeled points to the unlabeled points through the graph [31].

— *Label Spread (LS).* Is a semi-supervised learning algorithm that assigns labels to previously unlabeled data points. It is similar to Label Propagation (LP), but it uses a different way of updating the labels . LS preserves the initial labels of the labeled points and only updates the labels of the unlabeled points. LS also uses a renormalization trick to avoid numerical problems and ensure convergence [31].

— *Multilayer Perceptron (MLP).* It is a type of ANN that uses multiple layers of artificial neurons to learn features and patterns from data [32]. It is based on using a feedforward network that passes the input data through a series of hidden layers, each with a nonlinear activation function, and produces an output layer with a linear or nonlinear activation function.

2.2.4. Evaluation

The last step is to evaluate the accuracy of the algorithm. For this, statistical metrics identify when a model is correctly predicting. The following paragraphs describe some of the metrics most commonly used in our SLR.

Confusion Matrix. A confusion matrix is a visualization of the performance of the algorithm. [33] Each column of the matrix represents the items in a predicted class, and in each row, the items in their actual class. The diagonal of this matrix shows the correctly classified items: True Positive (TP) and True Negative (TN). TP are items that correctly predict their class (p) in their actual class (p'), while False Positives (FP) are items that are not classified in their correct class (p). The same goes for False Negatives (FN), items that should be classified as (p') but are classified as (n) and TN, items

		Prediction class		
		p	**n**	**total**
Actual class	**p′**	True Positive (TP)	False Negative (FN)	P′
	n′	False Positive (FP)	True Negative (TN)	N′
	total	P	N	

Table 1: Example of a Confusion Matrix

that correctly predict their class (n) in their actual class (n'). Table 1 is an example of a confusion matrix [33, 34].

Precision. Precision, or confidence [35], shows us whether the model correctly identified the items in their proper classes (Equation 1) [33, 36].

$$Precision = \frac{TP}{TP + FP} \tag{1}$$

Recall. Recall, or sensitivity [35], shows us the proportion of correctly predicted classes in the total of items that are actually positive (Equation 2) [33, 36].

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

F1-score. The F1-score, or the f-score, is the harmonic mean of both precision and recall (Equation 3) [37]. This metric shows how well an algorithm performs.

$$F_1\text{-}score = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{3}$$

Accuracy. Accuracy is the ratio of correctly classified items to the total of classified items (Equation 4) [33, 36].

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{4}$$

2.3. ML for Non-Functional Requirement Classification

Binkhonain and Zhao [16] reviewed state-of-the-art learning algorithms used to identify and classify NFRs. They evaluated 24 studies, published from 2007 to 2017. Their objective was to explore the processes used to identify and classify NFRs, define the most used algorithms, and how their performance was measured. As results, they highlighted three open challenges: *(1)* the lack of open-source labeled datasets; *(2)* the lack of standard definition, classification, and representation of NFRs; and *(3)* the lack of techniques to identity and select features to use as input to learning approaches. In our SLR, we also discussed these challenges and pointed out future directions based on recent works in the field. Although we show that there have been significant advances in the field, which is shown by the accuracy results reached by learning techniques, there is still a limited corpus of datasets available. Consequently, it is unclear how the learning accuracies can be generalized across other datasets. Additionally, we will discuss other current challenges in Section 4.5.

3. The Review Methodology

We followed the Kitchenham and Charters SLR guidelines [38] to systematically investigate the use of learning techniques for NFR classification. In this section, we present the SLR methodology that covers two main phases: *planning the review* and *conducting the review*.

3.1. Planning the Review

The need for a systematic review. The main goal of this SLR is to systematically investigate recent state-of-the-art approaches for the classification of NFR. A previous literature review [16] was already performed and published on this topic. However, this review selected studies from 2007 to 2017. In this context, our SLR aims to extend this previous review by selecting studies up to date, *i.e.*, published between 2018 and 2022. We compare the findings between both studies. Furthermore, we discuss what has mainly changed from the finding in [16] (before 2018) to nowadays (2018 to 2022) in this field. By means of this outcome, we can detect advances in this field and limitations in current approaches to properly suggest areas for further investigation.

The research questions. The goal of this SLR is to answer the following main research question (RQ): *What studies have recently been reported in the literature on learning NFRs? And what are the main advances and challenges based on the findings of a previous study [16] in the field?*

Thus, we derived the following five research questions:

RQ1 *What NFRs have been considered by state-of-the-art approaches?* In this RQ, we seek to answer which NFRs are mostly considered by the authors.

RQ2 *What are the datasets used to evaluate state-of-the-art approaches?* In this RQ, we seek to understand which datasets are used in the field, how they have been built, and their availability to the public. Thus, we aim to discuss their main features and their quality.

RQ3 *What learning techniques are adopted when learning NFRs?* In this RQ, we list all the learning techniques used in the literature. In addition, we investigate their strength, weakness, and popularity.

RQ4 *How are learning-based techniques validated?* In this RQ, we seek to define the evaluation metrics employed by each study and how accurate the proposed approaches are. Moreover, we present a cross-comparison of studies using similar approaches.

RQ5 *What are the main advances and challenges in the field?* In this RQ, we seek to list the advances in the field based on a previous SLR [16]. Therefore, we compare our findings with the findings from [16] regarding the types of NFRs, the most used datasets, the learning approaches used, and the accuracy of the approaches. The main goal is to point out what has changed in the field in the last few years and what is still challenging, and thus needs further research in the future.

These research questions will enable us to highlight tendencies and challenges in state-of-the-art approaches, comprehend the changes from the previous review, present our conclusions, and propose future works.

The review protocol. We searched for all relevant papers published up to September 28th 2022. The search process involved the use of 4 scientific digital libraries: ACM Digital Library, IEEE Xplore Digital Library, Science Direct, and Scopus. We used selected keywords in the abstract to restrict the

search only to the SLR scope. Then, we extracted the data from the papers and applied the selection criteria defined in the next step (see Section 3.2) to each study. Each paper not selected due to the Within/Out of Scope criteria has been later peer-reviewed to ensure correctness.

3.2. Conducting the Review

The search string. We initially defined three search terms ("machine learning", "non-functional requirements", and "classification") based on our prior knowledge of the subject. Then, we searched for synonyms for each term and refined the search string. The search string used was the following:

> ("machine learning" `OR` "prediction" `OR` "artificial intelligence") `AND` ("non-functional requirements" `OR` "non-functional properties" `OR` "quality attributes") `AND` ("classification" `OR` "identification" `OR` "detection" OR "classify")

Finally, we used the abstract filter keyword in the ACM Digital Library, IEEE Xplore, and Scopus to refine the search even further. As Science Direct does not support that type of filter directly in the search string, we used the 'Title, abstract, or author-specified keywords' in the Advanced Search section.

The selection criteria. We used the following three Inclusion Criteria (IC), which a paper must satisfy all, and four Exclusion Criteria (EC), which a paper must satisfy none:

- IC1 English Papers: The paper is available online and in English.
- IC2 Within of Scope: The paper should be about the classification of non-functional requirements.
- IC3 Last 5 years of research: To select the most current papers, we consider the publishing date from 2018 to 2022.
- EC1 Abstract: Introductions to special issues, workshops, tutorials, conferences, conference tracks, panels, poster sessions, as well as editorials and books.
- EC2 Short Papers: Short papers (less than or equal to 4 pages) and work-in-progress.

EC3 Out of Scope: Pure artificial intelligence papers or studies from other subjects, such as Medicine.

EC4 Secondary Studies: Secondary studies, such as literature reviews, comparative articles, articles that present lessons learned, position or philosophical papers without technical contribution, were not included in this review.

As shown in Figure 1, 148 candidate papers were selected: 13 from the *ACM Digital Library*, 29 from the *IEEE Xplore Digital Library*, 15 from *Science Direct*, and 91 from *Scopus*. Of these, 111 of the 148 articles were excluded due to non-satisfying the IC and (or) satisfying the EC, 12 duplicated papers were excluded, yielding a total of 22 primary papers selected for data extraction. Information concerning each paper is shown in Table 2.

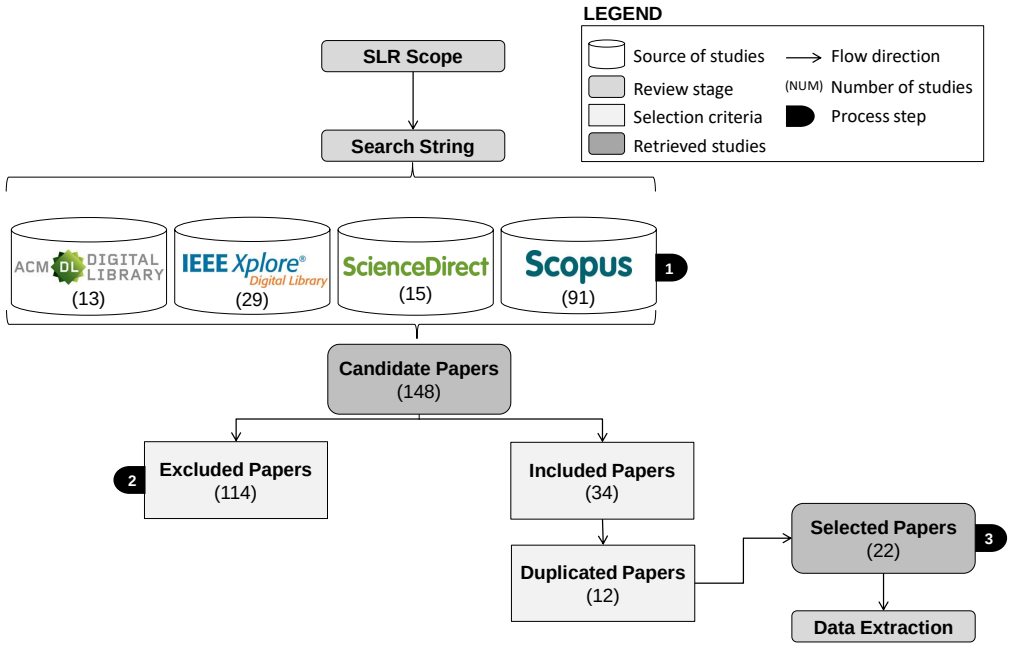

Figure 1: Flow of the paper selection process: papers retrieved from each digital library and details of the selection phases.

The data extraction. We extracted two types of data from each selected paper: bibliographic information of the study and data required to answer

research questions. Table 3 presents the bibliographic information data extraction sheet, and Table 4 presents the RQ data extraction sheet.

SLR data availability. Our Web supplementary material[4] provides the results of the search procedure from each of these steps.

Table 2: Reference list of selected studies

Study ID	Name	Reference
S1	Automated Identification of Security Requirements: A Machine Learning Approach	[6]
S2	Classifying non-functional requirements using RNN variants for quality software development	[7]
S3	A Systematic Approach of Dataset Definition for a Supervised Machine Learning Using NFR Framework	[8]
S4	Non-Functional Requirements Classification with Feature Extraction and Machine Learning: An Empirical Study	[39]
S5	Automatic Multi-class Non-Functional Software Requirements Classification Using Neural Networks	[9]
S6	CNN-BPSO approach to Select Optimal Values of CNN Parameters for Software Requirements Classification	[40]
S7	Identifying Non-functional Requirements from Unconstrained Documents using Natural Language Processing and Machine Learning Approaches	[10]
S8	Comparative Analysis for Predicting Non-Functional Requirements using Supervised Machine Learning	[41]
S9	Identification of Architecturally Significant Non-Functional Requirement	[42]
S10	A Pipeline for Automating Labeling to Prediction in Classification of NFRs	[11]
S11	Identifying Functional and Non-functional Software Requirements From User App Reviews	[43]
S12	Studying the commonalities, mappings and relationships between non-functional requirements using machine learning	[12]
S13	Framework for prediction and classification of non functional requirements: a novel vision	[13]

[4]https://github.com/pvliborio/slr-supplementary-material

Study ID	Name	Reference
S14	A Zero-Shot Learning Approach to Classifying Requirements: A Preliminary Study	[14]
S15	SABDM: A self-attention based bidirectional-RNN deep model for requirements classification	[15]
S17	Requirements and GitHub Issues: An Automated Approach for Quality Requirements Classification	[44]
S18	Software Requirements Classification using Machine Learning algorithm's	[45]
S19	Automatic Quality Attribute Scenarios Identification and Generation from Quality Attribute Requirements	[46]
S20	Mining non-functional requirements using machine learning techniques	[47]
S21	A Deep Learning-Based Framework for the Classification of Non-functional Requirements	[48]
S22	Software requirements classification using machine learning algorithms	[49]
S23	Study of the performance of various classifiers in labeling non-functional requirements	[50]

Table 3: Bibliographic information sheet

Data Item	Description
Database	The database from which the paper was extracted.
Author	List of authors of the paper.
Title	Title of the paper.
Book title	Title of the book or proceedings in which the paper was published.
Pages	Number of pages.
Year	Year of publication.
Publisher	The publisher of the paper.
DOI	DOI reference.

4. Results and Discussion of the Research Questions

In this section, we describe the findings for each research question defined in Section 3.

Table 4: RQs answering information sheet

Data Item	Description
Paper summary	A summary of the main purpose of the selected paper.
Dataset	The name and link of the dataset used.
NFRs	Types of NFRs identified (if any).
ML algorithm	Type of the ML algorithm used to identify or classify the textual requirements
Algorithm input	The input of the algorithm (*e.g.* labeled requirements specifications).
Evaluation methods	Detail of how the effectiveness of the algorithm was measured in the selected paper and what were the results.
Tools	The name of the tool(s) used (if any).
RQs	The research questions of the study, if defined.

4.1. Results RQ1 — most considered NFRs

To answer this question, we surveyed all the NFRs mentioned in the studies included in our review, classified them into 21 categories, populated the NFR elements of our information sheet described in Table 4, and counted their occurrences. The results are shown in Figure 2. Analyzing the figure illustrates that, for example, *Portability* appears in 9 studies while *Reliability* appears only in 4. The category 'Not Specified' was used for papers that did not specify the NFRs used for their classification. Moreover, the category 'It's Not NFR Specific' was used for papers that only classified if the input was or was not an NFR.

The NFRs most frequently considered were *Usability*, *Security*, *Performance*, and *Maintainability*, each appearing in at least 14 studies, suggesting that they are relatively well understood and commonly implemented by state-of-the-art approaches. Among the considered NFRs, the least frequently addressed were *Safety*, *Verifiability*, *Functional Suitability*, *Capability*, and *Efficiency*. These NFRs were mentioned in only one study each, suggesting that they may be less defined in existing datasets and rarely targeted by state-of-the-art approaches. In the following section, we will demonstrate that most studies rely on reusing state-of-the-art datasets that have already been labeled, highlighting the absence of datasets specifically designed to address these particular NFRs. In most existing datasets, these requirements are often treated as part of broader, higher-level requirements. For instance, *Efficiency* is typically considered within the scope of Performance, while *Ca-*

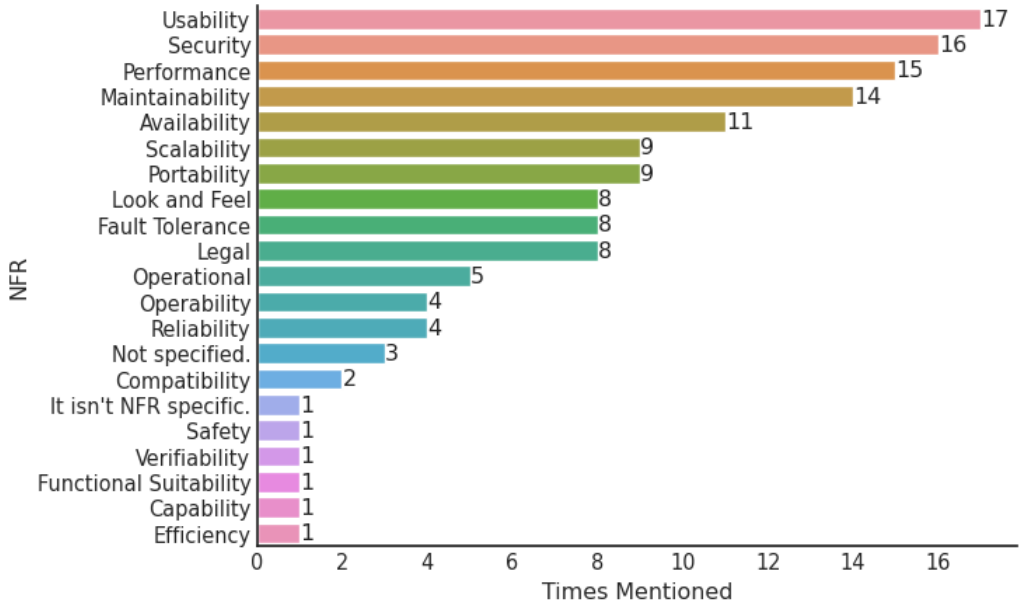

Figure 2: Number of occurrences of each category.

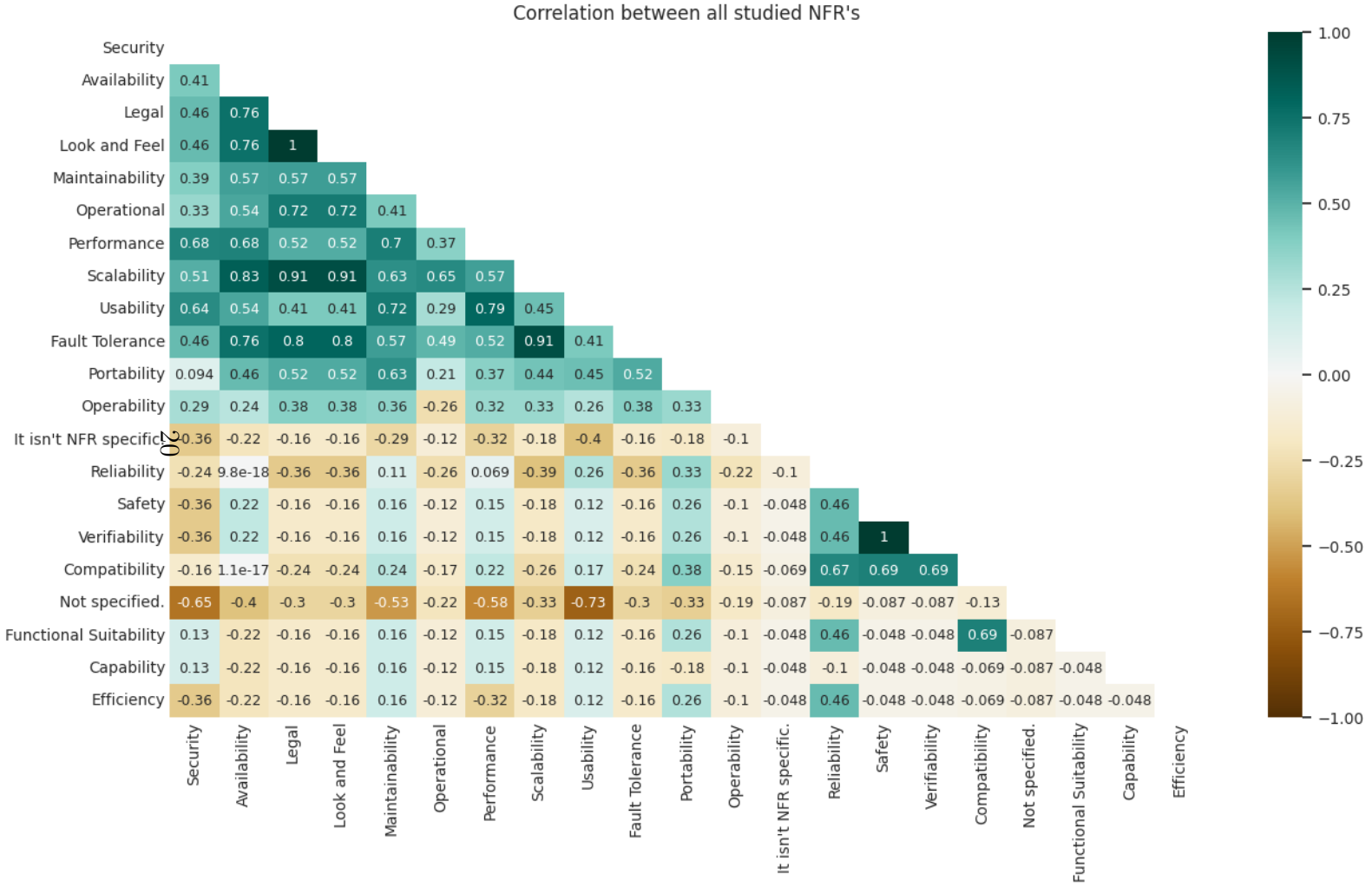

Figure 3: Correlation Triangle between all NFRs.

pability is considered under *Usability*.

We also performed a correlation analysis to identify which NFRs tend to cooccur in the same studies. The results are shown in Figure 3. Notice that we excluded the categories 'Not Specified' and 'It's Not NFR Specific'. The pair of NFRs that were the most strongly correlated are *Look and Feel* and *Legal*, along with *Safety* and *Verifiability*, both with a correlation coefficient of 1. This suggests that studies that address one of these NFRs are likely to also address the other in some way. On the other hand, the pair of NFRs least correlated, with some relevance, was *Scalability* and *Reliability*, with a correlation coefficient of -0.39. This implies that there is a weak negative relationship between *Scalability* and *Reliability*. However, this relationship is not very strong or consistent and does not imply causation. There may be other factors that influence both of them, or the correlation may be due to chance.

However, it was noted that some of these correlations may be biased by multiple studies using the same datasets, so future research to build new datasets with the consideration of less used NFRs is needed.

RQ1: *What NFRs have been considered by state-of-the-art approaches?*
Result: *The NFRs most frequently considered were* ***Usability***, ***Security***, ***Performance***, *and* ***Maintainability*** *each appearing in studies 17, 16, 15 and 14, respectively. However, it is due to the reuse of datasets by several selected studies.*

4.2. Results RQ2 — most used datasets

Similarly to the previous question, to answer this question, we surveyed all datasets that were used in the studies included in our review, populated the *Dataset* item in our information sheet with their names (if available) described in Table 4, and counted their total occurrences. In cases where papers use multiple datasets, computations are performed for each individual dataset used in the experiments. For example, S5 uses both the SecReq and PROMISE datasets. In Figure 4, their computations are represented by

the light orange and dark blue sections of the graph to indicate the analysis performed on each respective dataset. Sometimes, the same category appears more than once in a study, and we count it every time. This makes the total higher, but it helps us with some categories that are more broad, like the proprietary category. For those, we care more about the total number of datasets than the number per study (which would be 1 for all of them).

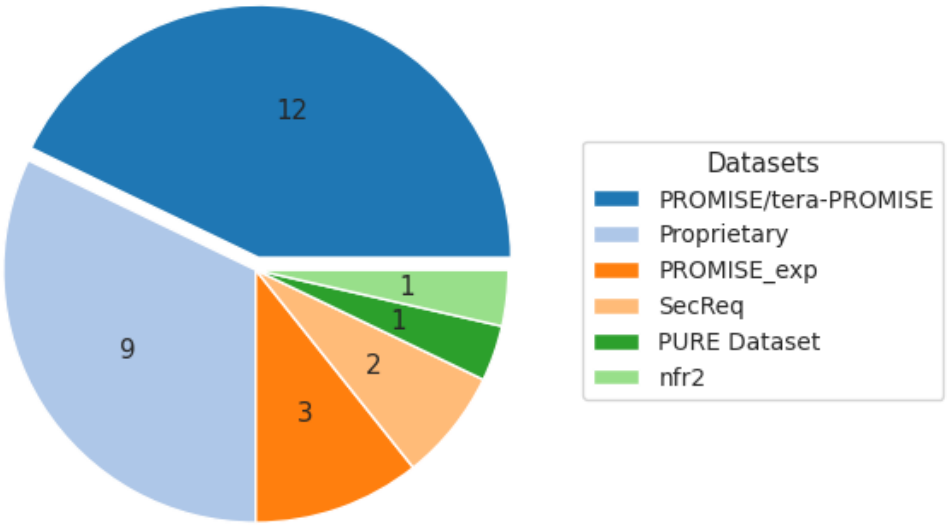

Figure 4: Number of occurrences of each dataset category.

The *PROMISE/tera-PROMISE* was the most used, appearing 12 times in all selected studies. Additionally, the *Proprietary* category, which groups multiple datasets, not all named, created by the authors during their study, was the second, appearing 9 times. However, it should be noted that 44% of all these occurrences came from S10, as it uses 4 different proprietary datasets.

The *PROMISE/tera-PROMISE* dataset accounted for almost half of all used datasets. One possible implication of this is that it is more accessible, reliable, or comprehensive than the other available datasets and that they cover a wide range of NFRs and software domains. Proprietary datasets are also largely used in the literature, however they are not available to the

community. Proprietary datasets may reflect the specific needs or preferences of the study or research group.

The least frequently used datasets were the *PURE Dataset* and *nfr2*, each appearing in one single study. This indicates that they may be less accessible, reliable, or comprehensive than the other datasets, or that they cover a narrow range of NFRs and software domains, or both. For example, the PURE dataset is a dataset of publicly available natural language requirements documents, which needs some preprocessing to be usable. These datasets may have limited applicability or generalizability to other contexts or scenarios.

These findings suggest that there is a lack of availability of datasets for classifying NFRs in software development and that some datasets may be more suitable or popular than others for certain types of NFRs or approaches, and to further analyze these initial findings, we will be discussing all categories in the next subsections.

4.2.1. PROMISE

The *PROMISE/tera-PROMISE*'s NFR dataset is a publicly accessible collection of natural language requirements for software systems, annotated as either functional or non-functional. The dataset comprises 625 requirements (255 functional and 370 non-functional) extracted from 15 projects. The NFRs are further categorized into different types, such as Availability (A); Fault Tolerance (FT); Legal (L); Look & Feel (LF); Maintainability (MN); Operational (O); Performance (PE); Portability (PO); Scalability (SC); Security (SE); and Usability (US). The dataset originated in the PROMISE dataset repository as part of the studies performed in [3] and [51]. It was hosted on 'promisedata.org' before being migrated to the Open Science tera-PROMISE repository over time and then archived on Zenodo under the name *nfr*. This dataset was also used in the *RE 2017 Data Challenge* under the name *Quality Attributes* Dataset. The copies at Zenodo [5] and at the RE 2017 Data Challenge site [6] are the only ones still available.

This dataset has been used in several studies (S2, S3, S4, S5, S6, S9, S10, S14, S15, S17, S20, S23), which attests to its credibility and validity in the academic community. It provides a rich and diverse source of data for training and testing different methods and models for NFR extraction

[5] https://zenodo.org/record/268542#.WkIdaN_ibIU

[6] http://ctp.di.fct.unl.pt/RE2017//pages/submission/data_papers/

and classification. Additionally, this dataset was created and maintained by collaborative platforms that facilitated its access and reuse. Furthermore, the widespread use and easy access of this dataset allow researchers to reproduce and compare the results of different studies in the current state of the art.

4.2.2. PROMISE_exp

The PROMISE_exp dataset is a public repository of software requirements and an expanded version of the PROMISE dataset created by [52]. The original PROMISE dataset contains 625 prelabeled software requirements, divided into functional and NFRs. NFRs are further classified into 11 types. The PROMISE_exp dataset contains 969 labeled software requirements obtained from a focused web search and manual extraction of documents containing software requirements records. The expansion aims to provide a larger and more diverse dataset for applying ML algorithms to software requirements classification tasks. The dataset is publicly available and has been evaluated using several ML algorithms, with some improvement in performance over the original dataset.

4.2.3. nfr2

The *nfr2* dataset is a publicly available dataset consisting of two subsets of text files. During our research, we found that it is identical to the data in the *PROMISE/tera-PROMISE* NFR dataset, labeled in the same manner, but formatted differently into a text file instead of an attribute-relation file. The first set contains 555 non-functional and functional requirements from the original dataset and a duplicate entry. The second set is a test set of 70 non-functional and functional requirements from the original dataset. It was used by only one study S8, which calls it *nfr2*, although it is simply named *Software Requirements Dataset* on Kaggle, its hosting platform.

We believe that this dataset has not been widely adopted because of the insufficient documentation of its data sources and annotation procedures. It does not cite or acknowledge its derivation from the *PROMISE/tera-PROMISE* NFR dataset, which we discovered by coincidence while examining both datasets simultaneously, thus raising doubts about its validity. Although its format may enhance readability for those unfamiliar with '.arff' files, its partition into two subsets is rather unbalanced and lacks an explicit methodology.

4.2.4. SecReq

Houmb et al. [53] and Schneider et al. [54] introduced *SecReq* and its dataset as part of an effort to allow reproducibility of their study results. It contains 510 requirements, made up of security-related requirements (187) and non-security-related requirements (323). The dataset is composed of three industry standards: Common Electronic Purse (ePurse) with 177 labeled entries, Customer Premises Network (CPN) with 124 labeled entries, and Global Platform Spec (GPS) with 210 labeled entries. The currently available dataset has two labels: 'sec' and 'non-sec'. However, the repository in Zenodo[7] also has a version of course with the combined labeling of five experts that shows different labels, *e.g.*, unknown, security-related, and permissive, which can be used in future research. This dataset was originally hosted at Hannover's Leibniz Universität, and a copy of it was used in the *RE 2017 Data Challenge*, but these resources are no longer available, so we cannot know for sure if there were also other versions of the tables for CPN and GPS.

The use of this dataset by other studies can be highly valuable, given its highly security-oriented labeling process. However, this dataset still needs resampling or even the consideration of additional stances in order to have a balanced dataset, as more than half of them are actually defined as 'non-sec'.

4.2.5. PURE Dataset

The PURE dataset is a publicly available dataset containing 79 requirement documents in different forms. It is described in [55] and can be used for Natural Language Processing (NLP) tasks such as ambiguity detection, requirement identification, and requirement categorization in different classes. The dataset contains 34,268 instances that cover multiple domains, and the size of the documents ranges from 7 to 288 A4 pages, with an average of 47 pages per document. The construction of the documents was divided into structure (S), unconstrained (U), and one statement (O). S7 defines that most documents are a combination of unconstrained content and one statement (about 38% of all documents), and the requirements are represented in one sentence. Structured documents were 15% of the documents.

The PURE dataset was used to perform supervised learning experiments on the classification of software requirements. To label the data, the re-

[7]Available at `https://zenodo.org/record/4530183`

searchers extracted instances from the requirements documents using a set of common criteria, such as: if it begins with a capital letter and concludes with a period, it is considered a sentence. To achieve this objective, these documents were preprocessed by converting them into XML format. Note that not all instances in a requirement document relate to a requirement sentence. To address this issue, the annotation process was conducted at the sentence level, and the extracted instances were manually labeled. To facilitate this process, an online website was developed called the *Requirements Classifier*, and a group of 43 software engineering experts were recruited to volunteer in the annotation process. The website was created solely for the purpose of conducting the study's classification process and no longer exists. Moreover, during the course of this study, we did not find any publicly available version of the resultant dataset, which would facilitate future research in the area.

The *PURE dataset* was created entirely from scratch by the authors. Only one study in our analysis undertook the construction of a completely new open-source dataset. The need for requirement annotation and preprocessing of the data may explain why most of the studies reuse state-of-the-art datasets.

4.2.6. Proprietary

In the course of our research, we encountered some datasets that were not publicly accessible, as they were created by the authors using private sources. This poses a challenge to the reproducibility and verifiability of the study by other researchers. However, these datasets still offer valuable insights into the research question. In the following part, we will describe the data collection process of the authors and explain the rationale for using a proprietary dataset for their specific study.

Firstly, the study S10 uses four datasets, from five sources. The sources comprised:

a. The tera-PROMISE repository with 625 labeled NFRs (see Section 4.2.1).

b. An open-source repository with 70 Software Requirements Specification (SRS) documents.

c. Three companies with a total of 30 SRS documents.

For the labeling of sources (b) and (c), they collaborated with three subject-matter experts who performed data extraction, cleaning, and labeling. They combined the data from (a) and (b) to form the Publicly Available Dataset (PAD) of 13,560 requirements, where 6,630 are NFRs. Although the authors give this name to the dataset due to the reuse of publicly available datasets from the literature, the resultant dataset is not available to the community.

They manually labeled 845 NFRs from PAD into 10 classes from the 11 existing in PROMISE. This manual labeling is used as input into the Snorkel tool[8] to automatically label the remaining 5,785 NFRs. For the validation of the automatic labeling process, they asked experts to label 200 out of 5,785 NFRs and compute the tool's accuracy. Thus, the training and validation sets are composed of a total of 845 plus 200 manual labeled NFRs.

They also extracted and classified NFRs from three business units into three industry-specific datasets (ISD-1, ISD-2, ISD-3). We conjecture that the authors created a proprietary dataset to address the need for more diverse data for the general classification since the Tera-PROMISE repository only contains requirements from 15 projects. The ISDs, which are mainly used for fine-tuning and for evaluation of their pre-trained model, are also useful for testing the model on domain-specific datasets that represent the use case for a real project.

In study S11 the authors developed a hybrid dataset of software requirements from two sources:

a. The PROMISE repository of labeled NFRs (see Section 4.2.1).

b. The user app reviews dataset consisting of 3,691 reviews from different apps in the Google Play Store and Apple's App Store.[9].

They labeled the requirements as functional (FR), non-functional (NFR), and non-requirement (NR) to form a common set of classes. They removed the *Fault Tolerance* and *Portability* NFR subcategories from the PROMISE dataset due to the low sample size and labeled the rest as NFR. The *Fault Tolerance* presents ten instances, and *Portability* presents only one in PROMISE, respectively. From the user app reviews source, they list 5 main categories: Feature Request, Bug Report, Problem Discovery, User

[8] `https://www.snorkel.org/`

[9] Publicly available at: https://data.mendeley.com/datasets/5fk732vkwr

Experience, and Rating. They labeled feature requests as FR and the other reviews as NR. Note that in the original paper, the authors did not mention how NFRs were labeled. However, the final number of NFRs labeled mentioned in the paper is 371 which did not correspond to the total number of NFRs from the PROMISE repository, which is 359. Thus, the resulting dataset has 12 additional instances compared to all the NFRs present in PROMISE after excluding the Fault Tolerance and Portability classes. It is questionable whether these 12 instances add enough value to classify NFRs in App reviews.

They balanced the dataset by downsampling the NR class to match the size of the NFR class, resulting in a dataset of 507 FR, 371 NR, and 371 NFR. Note that the data was not balanced with the number of FR. Since the objective of this study is to identify requirements from User App Reviews, which entails a rather specific task not much explored by academics, it justifies the need for constructing their own dataset. However, as mentioned, the authors did not label NFRs in app reviews, and this can be seen as a potential threat to the validity of their results.

The authors of S13 propose a framework for the classification and prediction of NFR. The framework has a Data Acquisition Layer, where they discuss different methods of acquiring data. They explain the pros and cons of each method. To test their framework, they collected a primary dataset from 312 IT professionals and academics. The dataset has 5,304 NFR instances, with 17 manually labeled NFRs.

The aim of S19, is to develop a model that can recognize the quality attribute scenario from the quality attribute requirements, specify the quality attribute, and define its scenario components. To achieve this, the authors constructed their own dataset of 891 quality attribute scenarios that were manually annotated using various sources such as architecture books, articles, and websites. They applied NLP techniques to preprocess the dataset. They then divided the dataset into 80% for training and 20% for testing.

In study S21, the authors start from a proprietary dataset derived from a previous study [56]. Comprising 1,484 instances taken from SRS documents. This corpus consists of five major classes of NFRs, including 480 instances with efficiency, 240 with maintainability, 156 with portability, 191 with reliability, and 417 with usability. From there, the authors define, and apply their own Easy Data Augmentation (EDA) method that uses a *"Sort & Concatenate"* strategy:

- They sort the instances from each class in ascending and descending order and concatenate them with the original instances.
- They remove the full stops from the first sentence and add white space between the instances.
- They assign the same class label to the combined sentence.

They claim that this technique preserves sentence structure, meaning and creates new requirements with the features of both instances (see Section 3 of S21 for a more detailed explanation of their proposed method). By applying it to the original dataset, they generate 2,976 new instances that are appended to the original, with the resultant dataset having 3,972 instances. The new dataset distribution consists of 1,440 sentence for efficiency, 720 sentence for maintainability, 468 sentence for portability, 573 sentence for reliability, and 1,251 sentence for usability, which shows a certain imbalance not dealt with in the study. The authors present a novel solution for creating the large annotated corpus necessary for neural network models. Therefore, this justifies their decision to build their own dataset.

In S23, the authors make use of 2 datasets for all their experiments:

- The tera-PROMISE repository of labelled NFRs (see Section 4.2.1).
- A dataset queried from Stack Overflow [10].

The second, which is proprietary, consists of a series of tagged posts that contain English text and sometimes code fragments. They specifically selected test- or performance-related tags. The resulting dataset contains 50,000 posts, with 20,166 posts corresponding to performance and 30,753 posts corresponding to test. Stack Overflow posts are multi-labeled, so 919 posts are labeled both testing and performance. They used both datasets in their own sets of experiments to compare results between a large corpus (Stack Overflow dataset) and a small corpus (tera-PROMISE dataset).

The purpose of their study was to test and compare the effectiveness of various ML methods for transforming natural language text into a suitable format for the applied algorithms. They also examined how the size of the

[10] `https://www.stackoverflow.com`

training corpus affected the performance of different techniques. To evaluate their approach to extracting useful data from natural language, they needed to create their own dataset.

The creation of a new dataset by the authors can be justified by the fact that the PROMISE_exp dataset was yet to be published while their study was conducted, so there were no instances of a larger public dataset of NFRs. They chose Stack Overflow as their source because it is a popular and public technical forum on software development.

RQ2: *What are the datasets used to evaluate state-of-the-art approaches?*
Result: *The most frequently used datasets were* ***PROMISE/tera-PROMISE*** *with 12 occurrences, together with the* ***Proprietary*** *category, with 9 occurrences. These two datasets represent more than half of the total number of sources used in all selected studies.*

4.3. Results RQ3 — learning techniques

Following the same main strategy as the questions previously answered, we surveyed all the learning algorithms considered in the selected studies, populated the *ML Algorithm* item in our information sheet with their names, and counted their occurrences per study. The results are shown in Figure 5.

Some algorithms were aggregated into a broader category due to similarity, such as the *Tree-based Algorithms* (TbA) category, which would have five different categories, otherwise.

Figure 5 shows that *NB* and *SVM* were the most frequently adopted learning techniques. These algorithms are known to be simple, fast, and effective for classification tasks. Alternatively, deep learning techniques such as *CNN*, *LSTM* and *Bi-LSTM* were less common, possibly due to the challenges of obtaining large labeled NFR datasets.

Next, we highlight some specificities of the employed approaches

- Due to the nature of the datasets previously presented (see Section 4.2), all implementations were employed together with some word encoding, or word embedding technique like 'Word2vec', for feature extraction.

- The implemented algorithms use well-known Python libraries, such as

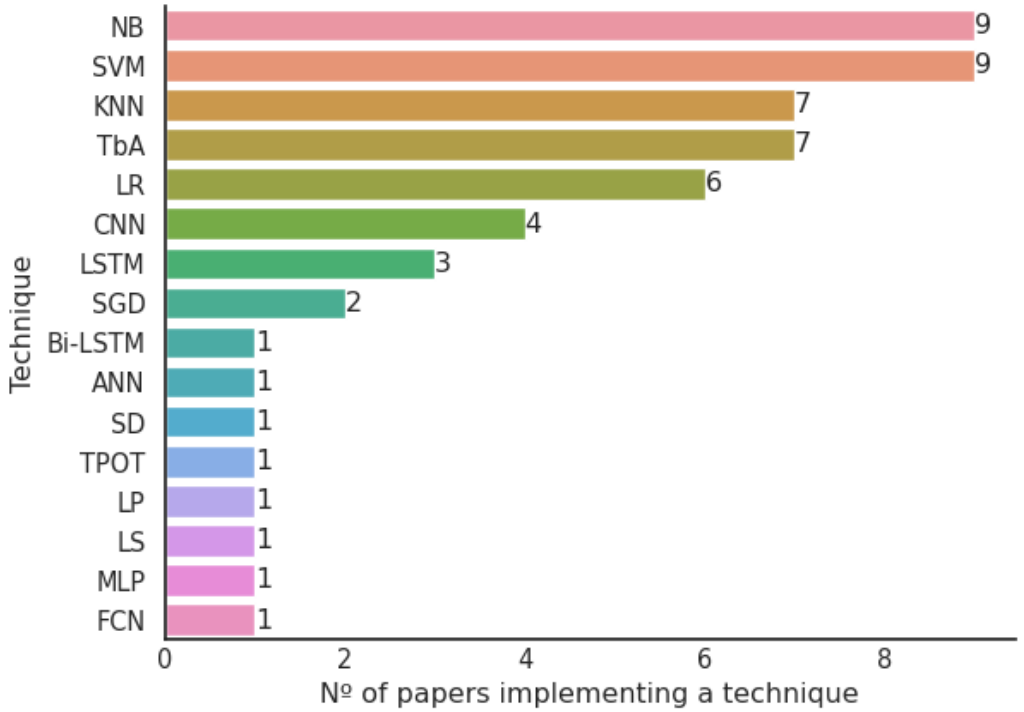

Figure 5: Most frequent algorithms.

Sickit[11], Keras[12], and Tensorflow[13].

In the following sections, we will dive more into the specifics of each technique, but refer to Section 2.2.3 for their definitions.

4.3.1. Naive Bayes (NB)

Used in 9 studies S1, S4, S8, S9, S12, S17, S22, S23, S7. Among the various versions studied of this algorithm, *Multinomial Naive Bayes* not only is it the most suitable for multiclass classification problems such as NFR classification, but it is also more implemented. NB can work well with sparse, noisy, or unbalanced data. However, it may not capture the nonlinear relationships between features, and it may assign zero probabilities to unseen

[11] `https://scikit-learn.org`
[12] `https://keras.io/`
[13] `https://www.tensorflow.org`

feature values.

4.3.2. Support Vector Machine (SVM)

Used in 9 studies S1, S4, S9, S11, S12, S22, S23, S7, and S17. SVM can handle complex data using functions such as Gaussian, polynomial, or RBF. SVM is accurate and general, but may be slow and memory intensive. In addition to that, SVC (Support Vector Classifier) is a type of SVM that specifically classifies data into different categories, as implemented in S17. Although, during our review, most of the implementations of this algorithm did not specify their kernel functions or subtypes, more details on the best implementations surveyed can be seen in Section 4.4.

4.3.3. K-Nearest Neighbors (KNN)

Used in 7 studies S1, S4, S8, S9, S12, S22, S23. KNN can handle non-linear data using different distance measures, such as Euclidean, Manhattan, or Minkowski. KNN is simple and accurate, but it can be slow and memory intensive when the data set is large. KNN is used for applications such as recommendation systems, pattern recognition, data mining, and more.

4.3.4. Logistic Regression (LR)

Used in 6 studies S1, S8, S12, S22, S23, S7. LR can handle binary or multiclass classification problems, and it can incorporate regularization terms to prevent overfitting. LR is fast and easy to interpret, but it may not capture the non-linear relationships between the features, and it may be sensitive to multicollinearity and outliers.

4.3.5. Tree-based Algorithms (TbA)

Used in 7 studies S1, S4, S8, S11, S12, S17, S23. This category comprises all surveyed tree-based algorithms like Fine Tree, Coarsed Tree, Decision Tree, Random Forest, Bagged Trees, RUSBoosted Tree, etc. Tree-Based Algorithms can handle binary or multiclass classification problems and can deal with categorical and numerical features. Tree-Based Algorithms are easy to understand and visualize.

4.3.6. Convolutional Neural Network (CNN)

Used in 4 studies S1, S5, S7, S21. CNN can handle complex and high-dimensional data, such as natural language. CNN can achieve state-of-the-art accuracy and generalization, but it may require a lot of training data

and computational resources. CNN may also be prone to overfitting and vanishing or exploding gradients.

4.3.7. Long Short Term Memory (LSTM/Bi-LSTM)

LSTM was used in 3 studies S1, S2, and S15. Bi-LSTM was used in only one, S15. Both versions of this algorithm are well suited for multiclass classification problems such as NFR classification. As will be explored in more detail in Section 4.4, this algorithm yielded one of the best results in all selected studies. However, LSTM/Bi-LSTM requires a lot of training data and computational resources and may also suffer from vanishing or exploding gradients and overfitting.

4.3.8. Stochastic Gradient Descent (SGD)

Used in 2 studies S3, S11. SGD can handle large and high-dimensional data sets efficiently, as it requires less computation and memory than batch gradient descent. SGD can also escape local minima and explore different solutions, as it introduces some randomness and noise in the optimization process. However, SGD can also have some drawbacks, such as being sensitive to the choice of learning rate and batch size, oscillating around the optimal solution, and converging to a suboptimal solution. SGD is widely used for applications such as linear and logistic regression, support vector machines, and neural networks.

4.3.9. Subspace Discriminant (SD)

Used only in 1 study S1. SD can handle binary or multiclass classification problems and can handle categorical and numerical features. SD is robust and accurate, but may require more computational resources and be harder to interpret.

4.3.10. Artificial Neural Network (ANN)

Used only in 1 study S5. ANN can achieve state-of-the-art accuracy and generalization but may require a large amount of training data and computational resources. ANN can also suffer from overfitting and vanishing or exploding gradients.

4.3.11. TPOT

Used only in 1 study S17. TPOT can handle binary or multiclass classification problems, and it can work well with large, high-dimensional datasets.

However, TPOT also has some weaknesses, as mentioned in their documentation[14]:

- TPOT can take a long time to run, depending on the size of the dataset. With the default settings, TPOT will evaluate 10,000 pipeline configurations before finishing.
- TPOT may not find the optimal pipeline, since TPOT may get stuck in local optima or converge to a suboptimal solution.
- TPOT may not be suitable for all types of problem, as it only supports classification and regression tasks, and it only uses scikit-learn models and methods.
- TPOT may not be easy to interpret or explain as it may generate complex pipelines with many steps and parameters that are difficult to understand or justify. TPOT may also lose some transparency and control over the modeling process, as it automates most of the decisions and choices.

4.3.12. Label Propagation (LP) and Label Spread (LS)

Used only in 1 study S23. LP and LS can handle binary or multiclass classification problems, and it can work well with sparse, noisy, or imbalanced data. However, LP may not capture the non-linear relationships between the features, and it may produce different results depending on the initial labels and the order of propagation.

4.3.13. Multilayer Perceptron (MLP)

Used only in 1 study S23. MLP can handle binary or multiclass classification problems and can handle categorical and numerical features. MLP is powerful and flexible, but may require a lot of training data and computational resources. MLP may also suffer from overfitting and vanishing or exploding gradients.

In summary, we observed that some studies used ensemble methods such as *Random Forest* and *TPOT* to combine multiple algorithms and improve

[14] `http://epistasislab.github.io/tpot/using/`

performance. Furthermore, some studies used semi-supervised learning techniques such as *Label Propagation* and *Label Spread* to leverage unlabeled data to learn NFR. Finally, we noticed that some studies used domain-specific algorithms such as *Subspace Discriminant* and *MLP* to capture the characteristics of NFRs in different domains.

RQ3: *What learning techniques are adopted when learning NFRs?*
Result: ***16*** *different learning techniques are adopted in the reviewed studies, being* ***Naive Bayes*** *and* ***SVM*** *the top-2 in usage, respectively.*

4.4. Results RQ4 — evaluation of techniques

Among the 22 studies analyzed, 19 of them provided results for at least one of the four primary evaluation measures: precision, recall, F1 score, and accuracy. Table 5 shows a comprehensive summary of the evaluation results obtained for each ML algorithm, highlighting the best performance achieved by each study. For example, we can see in the table that S1 implements eight algorithms: LSTM (84% accuracy), NB (53.5% accuracy), LR (49.2% accuracy), SVM (71% accuracy), CNN (80% accuracy), KNN (74.7% accuracy), SD (64.5% accuracy), and TbA (80% accuracy). All algorithms are employed to predict the *Security* NFR. The best accuracy (84%) is reached by LSTM. As we can see in the table, LSTM is implemented by three studies, still the best result of accuracy is reached by S1. However, it is important to note that S2 and S15 use the PROMISE/tera-PROMISE dataset to predict different NFRs, such as *Performance, Security* and *Usability*.

Table 5 shows us that Naive Bayes (NB), Support Vector Machine (SVM), and Tree-based Algorithms (TBA) were the most used algorithms, with 9 papers each. Of the studies that provided accuracy as a metric, those with accuracy equal to or above 80% are S1 and S15 in LSTM, S15 in Bi-LSTM, S9 in Naive Bayes (NB), S9 and S11 in Support Vector Machine (SVM), S1 in Convolutional Neural Network (CNN), S9 in K-Nearest Neighbors (KNN), S11 in Stochastic Gradient Descent (SGD), S1 and S11 in Tree-based Algorithms (TbA).

Table 5 shows us that NB and SVM were the most used algorithms, with 9 papers each. Of the studies that provided accuracy as a metric, those with accuracy equal to or above 80% are S1 and S15 in LSTM, S15 in Bi-LSTM, S9

in NB, S9 and S11 in SVM, S1 in CNN, S9 in KNN, S11 in SGD, S1 and S11 in TbA.

While S1 only used the SecReq dataset (see Section 4.2.4), both S9 and S15 used the PROMISE dataset (see Section 4.2.1) and another not specified dataset, and S11 uses a Proprietary dataset (see Section 4.2.6).

With respect to NFRs, S1 only classifies *Security* NFRs. S9 classifies 11 NFRs being *Usability, Performance, Maintainability, Portability, Reliability, Safety, Verifiability, Compatibility* and *Availability*, while S15 classifies only 5, *Capability, Maintainability, Performance, Security* and *Usability*. S11 is the only that are not NFR specific, classifying the instances in *FR*, *NFR* or *non requirement (NR)*.

Table 5 shows that SVM S9 has an F1-score of 0.9 and S23 measured 0.94 but in KNN, S23 got 0.79 while S9 had a better performance with 0.89. Both S9 and S23 used PROMISE/tera-PROMISE and a proprietary dataset. Notice that each study uses a different proprietary dataset, thus it can explain the difference in the reached accuracy results.

RQ4: *How are learning-based techniques validated?*
Result: *Learning-based techniques are commonly evaluated and validated using key metrics such as* ***Precision****,* ***Recall****,* ***F1-score*** *and* ***Accuracy****.*

4.5. Results RQ5 — main advances and challenges

We will now compare our findings from the previous questions (RQ1–RQ4) with the findings of a previous SLR [16].

4.5.1. Results RQ1 — most considered NFRs

Binkhonain and Zhao [16] did not directly assess the NFRs considered by the studies they reviewed; instead, they presented the categories identified in each study divided into three NFRs: *Security, Performance*, and *Usability* related. Our finding in Section 4.1 shows that *Security, Performance*, and *Usability* were the most considered NFRs in the selected studies. It highlights the importance of these NFRs. However, NFRs such as *Compatibility* and *Functional Compability* did not appear in [16]. Similarly, NFRs *Palm Operational* and *Lifecycle* did not appear in our review. However, the definitions of each NFR are lacking in [16] and our study. The majority of the studies selected in our review do not define the NFRs used, they use datasets

Table 5: Best evaluation of each algorithm for each paper it appears

Study ID	ML Alg.	Precision	Recall	F1-score	Accuracy (%)
S1					84
S2	LSTM	0.717	0.715	0.7	71.5
S15		0.8	0.72	0.74	80
S15	Bi-LSTM	0.95	0.96	0.96	96
S1					53.5
S4		0.55	0.54	0.52	66
S8		0.66115	0.66667		66.667
S9		0.87	0.91	0.89	87
S12	NB	0.507	0.512	0.444	
S17		0.2951	0.2968	0.2861	
S22		0.71	0.73	0.72	
S23		0.92	0.91	0.92	
S7		0.838	0.838	0.836	
S1					49.2
S8		0.75969	0.75		75
S12	LR	0.581	0.615	0.562	
S22		0.75	0.75	0.74	
S23		0.95	0.95	0.95	
S7		0.856	0.858	0.855	
S1					71
S4		0.66	0.61	0.61	76
S9		0.87	0.83	0.9	88
S11		0.927	0.896	0.91	82.6
S12	SVM	0.715	0.722	0.713	
S22		0.73	0.73	0.72	
S23		0.95	0.95	0.94	
S7		0.836	0.852	0.84	
S17		0.2185	0.2499	0.2143	
S1					80
S5	CNN	0.82 - 0.88	0.90 - 0.97	0.86 - 0.92	
S7		0.922	0.914	0.915	
S21		0.95	0.96	0.95	
S5	ANN	0.82 - 0.86	0.82 - 0.85	0.82 - 0.84	
S1					74.7
S4		0.62	0.48	0.5	69
S8		0.49978	0.5		50
	KNN				

Study ID	ML Alg.	Precision	Recall	F1-score	Accuracy (%)
S9		0.88	0.9	0.89	87
S12		0.616	0.63	0.616	
S22		0.66	0.66	0.66	
S23		0.8	0.8	0.79	
S1	SD				64.5
S3	SGD	0.75	0.97	0.84	
S11		0.91	0.916	0.913	88.3
S1					80
S4		0.18	0.17	0.15	51
S8		0.61278	0.60714		60.714
S23	TbA	0.91	0.91	0.91	
S11		0.861	0.887	0.873	80.2
S12		0.67	0.659	0.637	
S17		0.3751	0.3011	0.2291	
S17	TPOT	0.4406	0.3027	0.2409	
S23	LP	0.7	0.68	0.65	
S23	LS	0.7	0.67	0.65	
S23	MLP	0.38	0.66	0.36	
S23	FCN	0.96	0.94	0.95	

already published and labeled by experts in the field. We do search for the original studies of the datasets, but they may pose a threat to validity due to their lack of adherence to a strict validation protocol, as seen in more recent studies.

Most of the studies reviewed by us agreed with the majority of the NFRs because they used the same dataset, but even then some NFRs were excluded due to a lack of instances or relevance to their study. This shows us that there exists a path of improvement to future research.

4.5.2. Results RQ2 — most used datasets

The most used dataset was PROMISE/tera-PROMISE, a publicly available dataset of requirements from software systems created in 2005. In Binkhonain and Zhao [16], PROMISE/tera-PROMISE was also the most used dataset (see Section 5.3.1 of [16] for details). Concordia, the second most used open dataset, did not appear in our review. However, Concordia is no longer available, and we were unable to find any specifics about this dataset described in the literature.

Binkhonain and Zhao [16] found that the corpus of available datasets was

lacking and needed improvement to encourage researchers to conduct their experiments. The PROMISE/tera-PROMISE expansion, PROMISE_exp [52], shows that we are advancing in this challenge. However, we reinforce the results of the previous review, showing that the available datasets are very limited: (1) they are unbalanced; (2) they have very few labeled instances; (3) they lack a strict protocol about how experts are labeling such NFRs.

4.5.3. Results RQ3 — learning techniques

Regarding the algorithms used in the studies, NB and SVM are the most widely used in the literature, with 9 studies each. Both of them are also the top-2 used algorithms in the previous study [16]. Supervised learning algorithms are still the most used ML algorithm type in the literature, as none of the algorithms used in the studies we reviewed were unsupervised. More research on the use of other ML algorithm types, especially the unsupervised ones, is needed to verify the advances of these algorithms and their strengths and weaknesses in relation to supervised learning algorithms.

4.5.4. Results RQ4 — evaluation of techniques

Lastly, regarding the evaluation of the techniques, Precision, Recall, F1-score and Accuracy continue to be the most used metrics, as shown in Section 4.4 and the study of [16]. Newest studies performed better in both F1-score, the harmonic mean between Precision and Recall, and Accuracy related to the studies reviewed in [16].

RQ5: *What are the main advances and challenges in the field?*
Result: *Our review shows the advances in the field in comparison to the previous SLR, mainly regarding the* ***accuracy of learning techniques****. However, it is important to notice that the majority of the techniques are evaluated using* ***very specific datasets and NFRs****. Thus, there is still a long way to go.* ***Only one new open-source dataset*** *(PROMISE_exp) was proposed since and this dataset is not a completely new contribution, it is an expansion of a previous one with similar limitations of balance. The* ***use of unsupervised learning*** *in the classification of NFRs is an area to be better explored using the latest advancements in the field. In addition, the* ***lack of definition and standardization of NFRs*** *greatly hampers research in the field.*

Table 6: Summary of open challenges

Open Challenge	Consideration in Pratical Applications	Research Needed
Use of unsupervised ML approaches		
Use of Large Language Models		
Definition and standardization of NFRs		
Proposition of new datasets		
Support developers to address NFRs		
Transfer Learning across approaches.		

5. Emerging Research Themes

As we have shown, today there is very little experimental knowledge about the identification and classification of NFRs. The main limitation is the available resources in the community, such as new and robust datasets and standards about NFRs. Table 6 shows a summary of the open challenges that will be discussed in this section.

The use of other ML approaches. Most of the approaches reviewed in this study were supervised ones. The use of unsupervised ML approaches in the recent literature was not further explored since the previous study [16]. Also, new technologies, like Large Language Models (LLM) have been increasingly used for the most different tasks. Research on the possibility of their use in the identification and classification of NFRs is recommended.

Definition and standardization of NFRs. Managing the quality of software systems is essential for the efficient maintenance and evolution of their functionalities, saving computational and human resources. NFRs play a crucial role in this aspect. However, there is no consensus in the requirement engineering community on NFRs (see Section 2.1). There is also no list of the categories and subcategories of NFRs (see Section 4.1). This makes both the research and comparisons between them a difficult task. Therefore, the definition and standardization of NFRs are imperative to support further research in the area.

A repository of datasets.. The availability of a repository with all the datasets mentioned in this work is of great relevance to the field. Thus, we will make available a collection of publicly available datasets and other datasets as authorized by the authors of each work.

Transfer Learning across approaches. To do.

Proposition of new datasets. The existing datasets are still very limited regarding their context and domain. For example, PROMISE (see Section 4.2.1) is from 2006. Therefore, the creation of a new dataset is of extreme importance in the field. Thus, in future work, we plan to design and execute questionnaire studies with experts to label NFRs. Propose a catalog of good practices to label NFRs....

Support developers to address NFRs. The accomplishment of NFRs is important in different areas, such as games, artificial intelligence, the Internet of Things (IoT), and others. In this context, we will identify and mine representative datasets from public software repositories of different areas and domains. We plan to collect and analyze data about different versions of software systems over time. These data include change history, issue logs, commit and pull request information, and other relevant metrics. We will then conduct an experimental evaluation using ML to obtain quantitative knowledge about the evolution of the NFRs.

Such studies should provide qualitative insights into how (and why) certain NFRs are perceived as critical during the development of these systems. Examples of NFRs relevant to IoT systems include response latency, energy consumption, service availability, and security of transmitted data. Therefore, we need empirical knowledge to help us understand the improvement or deterioration of such NFRs.

The obtained results aim to provide relevant information to assist developers in making more informed decisions when optimizing the source code and architecture of the system, aiming to more efficiently meet the defined NFRs for the project. This can result in more robust, efficient, and reliable software systems.

With this project, we aim to identify, classify, and obtain a broad experimental view of the NFRs of systems, with detailed information regarding the impact of applied changes on the quality attributes of these systems.

6. Threats to Validity

In this section, we discuss the threats to the validity of this review. As in any other SLR, we may have been threatened by inaccuracy in the identification of studies, data extraction, or selection of studies based on inclusion/exclusion criteria. In the following, we will discuss the internal and external threats to validity [57]:

Internal validity. To mitigate as much as possible internal threats to the validity of our study, we performed our search in multiple databases. The results of that search were then reviewed by two of the authors. However, it is possible that studies were missed when none of the keywords used in the search string was present in the title or abstract of the paper.

The inclusion/exclusion process may also have been threatened. Some of the candidate papers were not clear about their objectives, which complicated the process and may have led to relevant studies being excluded and studies that should not have been selected. Also, each database has its own way of showing bibliographic data; the data extracted for bibliographic information could lead to an erroneous selection of studies. To minimize these risks, the process was reviewed by two of the authors after its completion. When a doubt arises, a discussion takes place, and a consensus is decided on whether the study should or should not be selected.

In some of the included articles, the information required to answer the research questions was not always obvious. To ensure validity, we analyzed other sources, including related studies. In case of doubt, the discussion procedure used in the inclusion/exclusion process was applied. However, we acknowledge that the data extraction of a paper is subjective.

External validity. The search string employed might have impacted the completeness of the selected studies. Our search could have potentially overlooked studies in which the authors used different keywords. To address and mitigate these limitations while ensuring replicability, this SLR used a rigorous protocol described in Section 3. The protocol was thoroughly discussed before the start of the review to improve the reliability of the selection and data extraction processes. However, we ackknowledge that it was designed considering the scope of our specific review and may limit the inclusion of studies from other domains.

The definition of open challenges in this study is also considered a potential threat to external validity. All authors are researchers in the software engineering field, and none of them are authors of the selected primary studies. Thus, we are not aware of any bias introduced during the analysis. However, there is the possibility that the subjectivity involved in the interpretation of open challenges was influenced by the author's personal opinions. Different researchers may identify additional open challenges that were not covered in this particular review and should be included in future works to provide a more comprehensive analysis.

7. Conclusion

In this study, we systematically reviewed 22 selected papers and their state-of-the-art approaches for the identification and classification of NFRs.

We defined five research questions related to (1) the NFRs considered by the studies, (2) the datasets used in the literature, (3) the learning techniques employed by each study, (4) how these learning techniques were validated, and (5) the main advances and challenges in the field. This study aims to point out the most efficient approaches that aid developers and researchers in identifying and classifying NFRs.

We concluded that while there are advances in the field compared to a previous study [16], there is still a long way to go. Most algorithms (11 out of 16) were used less than 5 times. Although the accuracy of the algorithms is promising for most of the algorithms (see Table 5), they are evaluated in specific contexts. An open-source robust dataset is needed for the advancement of research in the area and is recommended for future work.

Additionally, as future work, we plan to investigate open-source software repositories to establish a more efficient connection between expert skills and specific NFRs. Also, we aim to gain insights into how experts effectively address NFRs through the data mining and analysis of historical information of these repositories. This includes data such as change history, issue records, commit information, and pull requests. This research endeavor ultimately assists new developers in learning patterns of NFR resolution. Thus, avoiding the introduction of NFR issues in their software development processes through its identification, classification, and a guide for its resolution.

We also plan to establish a more robust definition for the group of most commonly used NFRs. The definition and standardization of NFRs have remained challenging, with limited progress since the previous study [16] By defining a standardized framework, we envision future works to become more cross-checkable, enabling the use of datasets from diverse sources for better comparison and analysis. Thus, our efforts in this field are directed toward fostering advancements in NFR understanding, and thus promoting consistency across different state-of-the-art studies.

Acknowledgments

This research was partially financed by Behring Foundation and the State Institute of Engineering and Architecture (IEEA), affiliated with the State Infrastructure Secretariat of Rio de Janeiro, through Contract 001/2021 with funding from the Government of the State of Rio de Janeiro.

References

[1] R. Young, The Requirements Engineering Handbook, Artech House professional development and technology management library, Artech House, 2004.

[2] M. Glinz, On non-functional requirements, in: 15th IEEE international requirements engineering conference (RE 2007), IEEE, 2007, pp. 21–26.

[3] J. Cleland-Huang, R. Settimi, X. Zou, P. Solc, The detection and classification of non-functional requirements with application to early aspects, in: 14th IEEE International Requirements Engineering Conference (RE'06), 2006, pp. 39–48. doi:10.1109/RE.2006.65.

[4] A. Aurum, C. Wohlin, Engineering and Managing Software Requirements, 2005. doi:10.1007/3-540-28244-0.

[5] L. Chung, J. C. S. do Prado Leite, On non-functional requirements in software engineering, Conceptual modeling: Foundations and applications: Essays in honor of john mylopoulos (2009) 363–379.

[6] A. Kobilica, M. Ayub, J. Hassine, Automated identification of security requirements: A machine learning approach, in: Proceedings of the 24th International Conference on Evaluation and Assessment in Software Engineering, EASE '20, Association for Computing Machinery, New York, NY, USA, 2020, p. 475–480. doi:10.1145/3383219.3383288.

[7] M. A. Rahman, M. A. Haque, M. N. A. Tawhid, M. S. Siddik, Classifying non-functional requirements using rnn variants for quality software development, in: Proceedings of the 3rd ACM SIGSOFT International Workshop on Machine Learning Techniques for Software Quality Evaluation, MaLTeSQuE 2019, Association for Computing Machinery, New York, NY, USA, 2019, p. 25–30. doi:10.1145/3340482.3342745.

[8] M. Marinho, D. Arruda, F. Wanderley, A. Lins, A systematic approach of dataset definition for a supervised machine learning using nfr framework, in: 2018 11th International Conference on the Quality of Information and Communications Technology (QUATIC), 2018, pp. 110–118. doi:10.1109/QUATIC.2018.00024.

[9] C. Baker, L. Deng, S. Chakraborty, J. Dehlinger, Automatic multi-class non-functional software requirements classification using neural networks, in: 2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC), Vol. 2, 2019, pp. 610–615. doi:10.1109/COMPSAC.2019.10275.

[10] Q. A. Shreda, A. A. Hanani, Identifying non-functional requirements from unconstrained documents using natural language processing and machine learning approaches, IEEE Access (2021) 1–1doi:10.1109/ACCESS.2021.3052921.

[11] R. Chatterjee, A. Ahmed, P. Rose Anish, B. Suman, P. Lawhatre, S. Ghaisas, A pipeline for automating labeling to prediction in classification of nfrs, in: 2021 IEEE 29th International Requirements Engineering Conference (RE), 2021, pp. 323–323. doi:10.1109/RE51729.2021.00036.

[12] A. M. Alashqar, Studying the commonalities, mappings and relationships between non-functional requirements using machine learning, Science of Computer Programming 218 (2022) 102806. doi:10.1016/j.scico.2022.102806.

[13] N. Handa, A. Sharma, A. Gupta, Framework for prediction and classification of non functional requirements: a novel vision, Cluster Computing 25 (2) (2022) 1155–1173. doi:10.1007/s10586-021-03484-0.

[14] W. Alhoshan, L. Zhao, A. Ferrari, K. J. Letsholo, A zero-shot learning approach to classifying requirements: A preliminary study, in: V. Gervasi, A. Vogelsang (Eds.), Requirements Engineering: Foundation for Software Quality, Springer International Publishing, Cham, 2022, pp. 52–59. doi:10.1007/978-3-030-98464-9_5.

[15] K. Kaur, P. Kaur, Sabdm: A self-attention based bidirectional-rnn deep model for requirements classification, Journal of Software: Evolution and Process n/a (n/a) (2022) e2430. doi:https://doi.org/10.1002/smr.2430.

[16] M. Binkhonain, L. Zhao, A review of machine learning algorithms for identification and classification of non-functional requirements, Expert

Systems with Applications: X 1 (2019) 100001. doi:10.1016/j.eswax.2019.100001.

[17] Ieee standard glossary of software engineering terminology, IEEE Std 610.12-1990 (1990) 1–84doi:10.1109/IEEESTD.1990.101064.

[18] B. Nuseibeh, S. Easterbrook, Requirements engineering: A roadmap, in: Proceedings of the Conference on The Future of Software Engineering, ICSE '00, Association for Computing Machinery, New York, NY, USA, 2000, p. 35–46. doi:10.1145/336512.336523.

[19] P. Shankar, B. Morkos, D. Yadav, J. Summers, Towards the formalization of non-functional requirements in conceptual design, Research in Engineering Design 31 (2020) 449–469. doi:10.1007/s00163-020-00345-6.

[20] P. P. Shinde, S. Shah, A review of machine learning and deep learning applications, in: 2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA), 2018, pp. 1–6. doi:10.1109/ICCUBEA.2018.8697857.

[21] Z.-H. Zhou, Machine learning, Springer Nature, 2021.

[22] P. Harrington, Machine learning in action, Simon and Schuster, 2012.

[23] C. Janiesch, P. Zschech, K. Heinrich, Machine learning and deep learning, Electronic Markets 31 (3) (2021) 685–695.

[24] A. C. Müller, S. Guido, Introduction to machine learning with Python: a guide for data scientists, " O'Reilly Media, Inc.", 2016.

[25] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016, http://www.deeplearningbook.org.

[26] K. Kaur, P. Kaur, Bert-cnn: Improving bert for requirements classification using cnn, Procedia Computer Science 218 (2023) 2604–2611, international Conference on Machine Learning and Data Engineering. doi:https://doi.org/10.1016/j.procs.2023.01.234.
URL https://www.sciencedirect.com/science/article/pii/S187705092300234X

[27] M. Schuster, K. Paliwal, Bidirectional recurrent neural networks, IEEE Transactions on Signal Processing 45 (11) (1997) 2673–2681. doi:10.1109/78.650093.

[28] B. Ghojogh, M. Crowley, Linear and quadratic discriminant analysis: Tutorial (2019). arXiv:1906.02590.

[29] R. S. Olson, J. H. Moore, TPOT: A Tree-Based Pipeline Optimization Tool for Automating Machine Learning, Springer International Publishing, Cham, 2019, pp. 151–160. doi:10.1007/978-3-030-05318-5_8.
URL https://doi.org/10.1007/978-3-030-05318-5_8

[30] W. Alhoshan, A. Ferrari, L. Zhao, Zero-shot learning for requirements classification: An exploratory study, Information and Software Technology 159 (2023) 107202. doi:https://doi.org/10.1016/j.infsof.2023.107202.
URL https://www.sciencedirect.com/science/article/pii/S0950584923000563

[31] X. Zhu, Z. Ghahramani, Learning from labeled and unlabeled data with label propagation (07 2003).

[32] P. Caceres, Introduction to neural network models of cognition - the multilayer perceptron.
URL https://com-cog-book.github.io/com-cog-book/features/multilayer-perceptron.html

[33] M. Vakili, M. Ghamsari, M. Rezaei, Performance analysis and comparison of machine and deep learning algorithms for iot data classification, CoRR abs/2001.09636 (2020). arXiv:2001.09636, doi:10.48550/arXiv.2001.09636.

[34] T. Fawcett, An introduction to roc analysis, Pattern Recognition Letters 27 (8) (2006) 861–874, rOC Analysis in Pattern Recognition. doi:10.1016/j.patrec.2005.10.010.

[35] M. Gharib, A. Bondavalli, On the evaluation measures for machine learning algorithms for safety-critical systems, in: 2019 15th European Dependable Computing Conference (EDCC), 2019, pp. 141–144. doi:10.1109/EDCC.2019.00035.

[36] S. Z. Mishu, S. M. Rafiuddin, Performance analysis of supervised machine learning algorithms for text classification, in: 2016 19th International Conference on Computer and Information Technology (ICCIT), 2016, pp. 409–413. doi:10.1109/ICCITECHN.2016.7860233.

[37] Y. Sasaki, The truth of the f-measure, Teach Tutor Mater (01 2007).

[38] B. Kitchenham, S. Charters, Guidelines for performing systematic literature reviews in software engineering 2 (01 2007).

[39] M. A. Haque, M. Abdur Rahman, M. S. Siddik, Non-functional requirements classification with feature extraction and machine learning: An empirical study, in: 2019 1st International Conference on Advances in Science, Engineering and Robotics Technology (ICASERT), 2019, pp. 1–5. doi:10.1109/ICASERT.2019.8934499.

[40] M. Bisi, K. Keskar, Cnn-bpso approach to select optimal values of cnn parameters for software requirements classification, in: 2020 IEEE 17th India Council International Conference (INDICON), 2020, pp. 1–6. doi:10.1109/INDICON49873.2020.9342381.

[41] V. Mir Khatian, Q. Ali Arain, M. Alenezi, M. Owais Raza, F. Shaikh, I. Farah, Comparative analysis for predicting non-functional requirements using supervised machine learning, in: 2021 1st International Conference on Artificial Intelligence and Data Analytics (CAIDA), 2021, pp. 7–12. doi:10.1109/CAIDA51941.2021.9425236.

[42] E. Mohammed, E. Alemneh, Identification of architecturally significant non-functional requirement, in: 2021 International Conference on Information and Communication Technology for Development for Africa (ICT4DA), 2021, pp. 24–29. doi:10.1109/ICT4DA53266.2021.9672235.

[43] D. Dave, V. Anu, Identifying functional and non-functional software requirements from user app reviews, in: 2022 IEEE International IOT, Electronics and Mechatronics Conference (IEMTRONICS), 2022, pp. 1–6. doi:10.1109/IEMTRONICS55184.2022.9795770.

[44] J. M. Pérez-Verdejo, Á. J. Sánchez-García, J. O. Ocharán-Hernández, E. Mezura-Montes, K. Cortes-Verdin, Requirements and github issues:

An automated approach for quality requirements classification, Programming and Computer Software 47 (2021) 704–721. doi:10.1134/S0361768821080193.

[45] G. Y. Quba, H. Al Qaisi, A. Althunibat, S. AlZu'bi, Software requirements classification using machine learning algorithm's, in: 2021 International Conference on Information Technology (ICIT), 2021, pp. 685–690. doi:10.1109/ICIT52682.2021.9491688.

[46] A. Tessema, E. Alemneh, Automatic quality attribute scenarios identification and generation from quality attribute requirements, in: 2021 International Conference on Information and Communication Technology for Development for Africa (ICT4DA), 2021, pp. 107–112. doi:10.1109/ICT4DA53266.2021.9672247.

[47] R. Jindal, R. Malhotra, A. Jain, A. Bansal, Mining non-functional requirements using machine learning techniques, e-Informatica Software Engineering Journal 15 (1) (2021). doi:10.37190/E-INF210105.

[48] M. Sabir, E. Banissi, M. Child, A deep learning-based framework for the classification of non-functional requirements, in: Á. Rocha, H. Adeli, G. Dzemyda, F. Moreira, A. M. Ramalho Correia (Eds.), Trends and Applications in Information Systems and Technologies, Springer International Publishing, Cham, 2021, pp. 591–601. doi:10.1007/978-3-030-72651-5_56.

[49] E. Dias Canedo, B. Cordeiro Mendes, Software requirements classification using machine learning algorithms, Entropy 22 (9) (2020). doi:10.3390/e22091057.

[50] L. Tóth, L. Vidács, Comparative study of the performance of various classifiers in labeling non-functional requirements, Information Technology and Control 48 (3) (2019) 432–445. doi:10.5755/j01.itc.48.3.21973.

[51] J. Cleland-Huang, R. Settimi, X. Zou, P. Solc, Automated classification of non-functional requirements, Requirements Engineering 12 (2) (2007) 103–120. doi:10.1007/s00766-007-0045-1.

[52] M. Lima, V. Valle, E. a. Costa, F. Lira, B. Gadelha, Software engineering repositories: Expanding the promise database, in: Proceedings of

the XXXIII Brazilian Symposium on Software Engineering, SBES '19, Association for Computing Machinery, New York, NY, USA, 2019, p. 427–436. doi:10.1145/3350768.3350776.

[53] S. H. Houmb, S. Islam, E. Knauss, J. Jürjens, K. Schneider, Eliciting security requirements and tracing them to design: an integration of common criteria, heuristics, and umlsec, Requirements Engineering 15 (2010) 63–93. doi:10.1007/s00766-009-0093-9.

[54] K. Schneider, E. Knauss, S. Houmb, S. Islam, J. Jürjens, Enhancing security requirements engineering by organizational learning, Requirements Engineering 17 (2012) 35–56. doi:10.1007/s00766-011-0141-0.

[55] A. Ferrari, G. O. Spagnolo, S. Gnesi, Pure: A dataset of public requirements documents, in: 2017 IEEE 25th International Requirements Engineering Conference (RE), 2017, pp. 502–505. doi:10.1109/RE.2017.29.

[56] M. Sabir, C. Chrysoulas, E. Banissi, Multi-label classifier to deal with misclassification in non-functional requirements, in: Trends and Innovations in Information Systems and Technologies: Volume 1 8, Springer, 2020, pp. 486–493.

[57] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, A. Wesslén, Experimentation in software engineering, Springer Science & Business Media, 2012.