

15. Practical Guidelines for the Selection and Evaluation of Natural Language Processing Techniques in Requirements Engineering

Mehrdad Sabetzadeh¹ and Chetan Arora²

¹ University of Ottawa, ON, Canada.

² Monash University, Vic, Australia.

Contributing authors: m.sabetzadeh@uottawa.ca;
chetan.arora@monash.edu;

Abstract

[Context and Motivation] Natural Language Processing (NLP) is now a cornerstone of requirements automation. One compelling factor behind the growing adoption of NLP in Requirements Engineering (RE) is the prevalent use of natural language (NL) for specifying requirements in industry. NLP techniques are commonly used for automatically classifying requirements, extracting important information, e.g., domain models and glossary terms, and performing quality assurance tasks, such as ambiguity handling and completeness checking. With so many different NLP solution strategies available and the possibility of applying machine learning alongside, it can be challenging to choose the right strategy for a specific RE task and to evaluate the resulting solution in an empirically rigorous manner. **[Content]** In this chapter, we present guidelines for the selection of NLP techniques as well as for their evaluation in the context of RE. In particular, we discuss how to choose among different strategies such as traditional NLP, feature-based machine learning, and language-model-based methods. **[Contribution]** Our ultimate hope for this chapter is to serve as a stepping stone, assisting newcomers to NLP4RE in quickly initiating themselves into the NLP technologies most pertinent to the RE field.

1 Introduction

NLP’s role in requirements automation is pivotal, due to the widespread use of natural language (NL) in industrial requirements specifications. Historically, NL has posed challenges for requirements analysis because of its inherent proneness to defects such as incompleteness and ambiguity. Recent breakthroughs in NLP, e.g., the emergence of large language models, have nonetheless drastically enhanced our ability to automatically analyze textual information. This development is poised to even further amplify the adoption and influence of NL in requirements engineering.

Due to the rapid advancement of NLP, newcomers to NLP4RE may feel overwhelmed by the numerous potentially applicable technologies. Another challenge is the necessity to empirically assess a proposed automation solution, ensuring proper optimization, and, where applicable, improve performance over existing solutions.

Over the past several years, we have studied various requirements automation problems, including checking compliance with requirements templates [5], glossary construction [7], model extraction [6], requirements demarcation [2], ambiguity handling [19], and question answering [20]. With the benefit of hindsight, this chapter aims to reflect on our research process and offer our collective insights into how we approach NLP4RE problems. Before we start, we need to emphasize that our perspective is *retrospective*. Given the fast pace of progress in NLP technologies, new considerations may surface, and existing technologies could become outdated. Therefore, it is important for readers to consider the time this chapter was written (2023) when dealing with new technologies. This advice applies to most works in the fast-changing field of Applied AI.

Structure. Section 2 outlines the steps for automating pre-processing, analysis, and post-processing in NLP4RE. Section 3 describes various NLP techniques and discusses their key considerations for automation in RE. Finally, Section 4 summarizes the chapter and presents conclusions.

2 Automation Steps in NLP4RE

The automation process for NL requirements and related textual artifacts can be structured into three sequential steps. These steps, shown in Figure 1 are: (1) Pre-processing, (2) Analysis, and (3) Post-processing. We outline these steps next.

2.1 Pre-processing

The goal of pre-processing is to automatically examine the NL content of requirements or requirements-related artifacts (e.g., design documents and code) and generate structured information for use by the Analysis step (Section 2.2).

The specific information targeted by pre-processing depends on the needs of the subsequent Analysis step. Importantly, the information to obtain through pre-processing relies on the selection of the *units of analysis* for the Analysis step. In the context of NLP, “unit of analysis” refers to the particular textual components that are intended for processing and interpretation. Some common units of analysis are: words, phrases, sentences, and paragraphs. For instance, when the objective of Analysis is

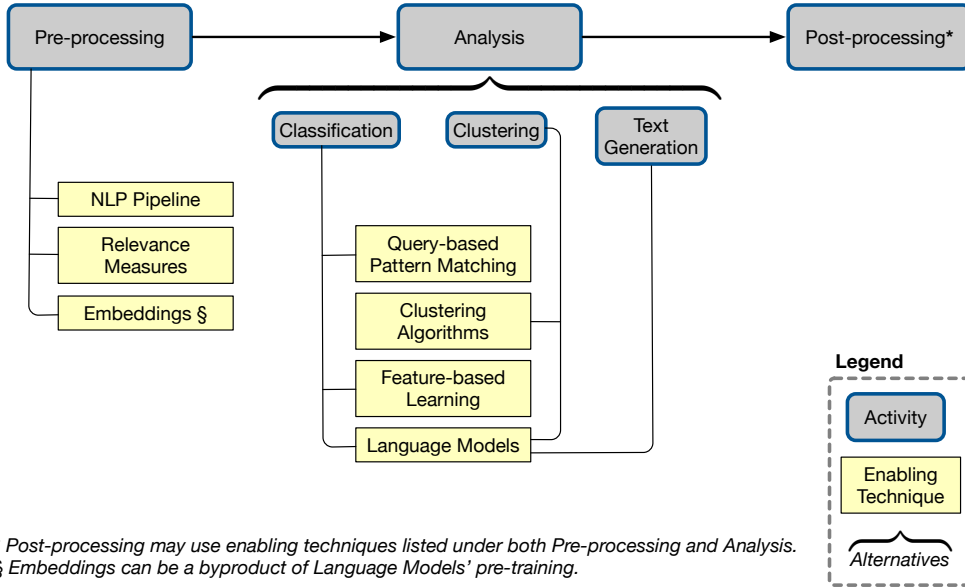


Fig. 1 NLP4RE Steps

identifying statements that contain requirements, sentences are frequently adopted as the units of analysis [2]. We note that there is a composition relationship between different unit types in NLP. For example, a sentence is made up of phrases, and a phrase is made up of words. Because of this characteristic, it is possible to use a combination of units of analysis simultaneously, e.g., phrases alongside sentences.

Typically, during the pre-processing phase, a set of “features” are calculated for the units of analysis (or the relationships between the units). A *feature* refers to a distinct attribute or characteristic of a unit of analysis. Features can be numeric (e.g., the number of tokens appearing in a sentence) or categorical (e.g., part-of-speech tags).

The most common enabling technologies for computing features are presented in Figure 1 under the Pre-processing activity. These technologies include the NLP Pipeline, Relevance Measures and Embeddings, which are discussed further in Section 3.

2.2 Analysis

The core of the process shown in Figure 1 is the Analysis step. In NLP4RE, this step typically manifests as one of the following three alternative activities: Classification, Clustering, and Text Generation.

2.2.1 Analysis Activities

Classification involves the assignment of labels or categories to the units of analysis. There are numerous NLP4RE use cases for classification. An example use case would be differentiating between functional (F) and non-functional requirements (NF) [27].

This task can be framed as the assignment of F and NF labels to the units of analysis, which in this context, are typically sentences within a requirements document.

Classification further extends to encompass *verbatim information extraction*. Verbatim information extraction involves directly extracting exact segments from source documents without abstraction, inference, or interpretation. This is done by marking off the text segments of interest and assigning labels to them. A typical use case is identifying requirements-related text segments in legal documents and annotating them with labels such as “permission”, “obligation”, “condition”, and “exception” [43].

Clustering leverages inherent similarities among the units of analysis to organize them into groups or themes. Unlike classification, which requires predetermined labels, clustering emphasizes the intrinsic structure and similarities within the content. The goal is to bring together similar content based on shared features, thereby avoiding the need for explicit predefined categories. This makes clustering particularly useful when dealing with problems where the labels are not well-defined or when exploring content where the underlying patterns might not be immediately evident. For instance, clustering can be used to identify groups of closely related requirements phrases during glossary construction [7]; here there is no predetermined set of classes for the groups of related phrases that will emerge.

Text Generation involves the automated creation of human-readable text based on either structured or unstructured inputs to aid the derivation, completion, understanding and communication of requirements. NLP4RE solutions based on text generation are a relatively recent development but are rapidly gaining momentum thanks to advances in generative language models like GPT [9]. Example use cases for text generation include constructing requirements models based on prompts and early-stage descriptions [16], summarizing requirements-related documents [26], and providing predictive assistance for requirements completion [31].

2.2.2 Technique Selection

Selecting suitable enabling technique(s) for the Analysis step is crucial. To make this selection easier, we have developed a decision process, shown in Figure 2. This simple process, which is based on our past experience, aims to facilitate narrowing the options for the Analysis step.

The first and most important criterion in this process is decision node (a), as depicted in Figure 2. This decision concerns whether we have a well-established and pre-existing set of conceptual categories relevant to automation. For instance, consider the task of classifying functional and non-functional requirements. In this task, the categories would be functional and non-functional, and this understanding exists before classification. For another example, consider the task of domain model extraction. Here, all pertinent categories are identified and can be listed, such as class, attribute, association, cardinality constraint, and so on. In contrast, consider a problem like identifying similar and potentially redundant requirements. When presented with a requirements document, predicting the number of equivalence classes (clusters) is impossible. As a result, there is no predefined set of conceptual categories for this problem that can be known beforehand.

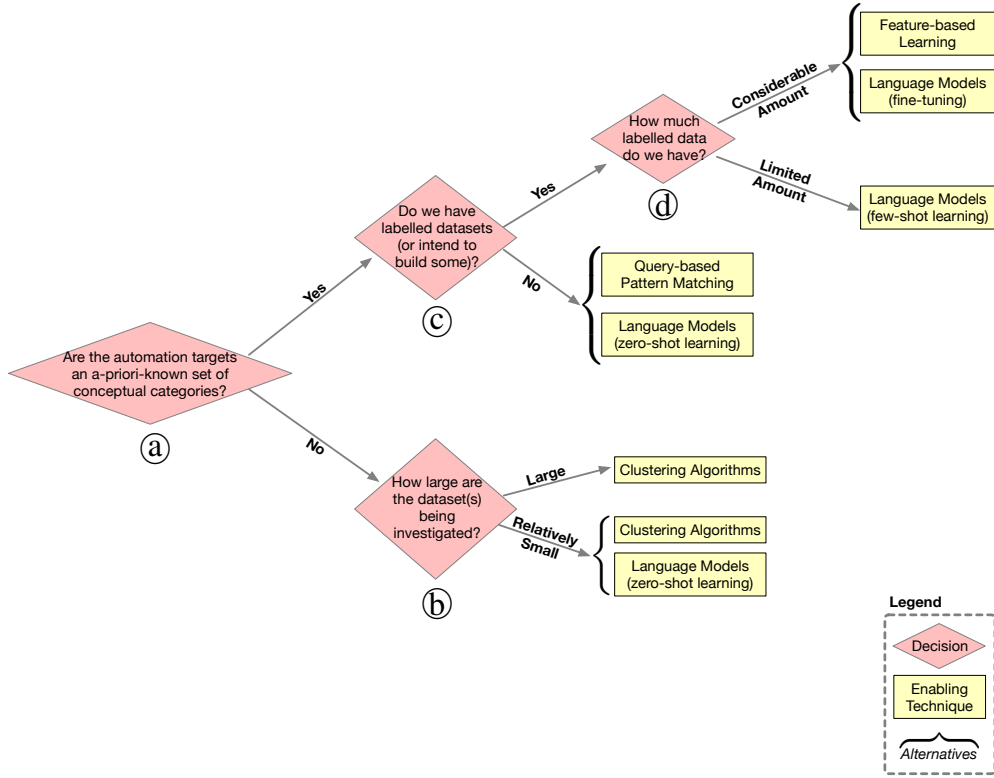


Fig. 2 Identifying Suitable Enabling Technique(s) for a Specific Analysis Task

Without established conceptual categories, dataset size is the next crucial criterion to consider. Historically, when predefined categories are not possible, clustering algorithms have been the preferred enabling technique for analysis. Recent advances in generative language models like ChatGPT and Llama nonetheless offer an alternative. For instance, when dealing with data sources such as requirements documents, one can use exploratory prompts like the following to uncover the main themes: “*Cluster the primary concepts in the following: [contents of the data source]*”.

As of writing, using a generative language model as an alternative to clustering algorithms is effective only for relatively small content volumes. Existing language models set a limit on total input and output tokens during interactive dialogs, known as “token limit” or “session size.” This limit defines how much context the model is capable of taking into account for response generation. Currently, the largest token limit we are aware of is 32,768 tokens (GPT-4-32k); this would be insufficient for many NLP4RE problems, such as traceability retrieval, which can potentially involve millions of tokens.

When an a-priori-known set of conceptual categories exists, the next question is whether a labelled dataset is available. This is captured by decision node (c) in Figure 2. When a labelled dataset is lacking, the most common enabling technique

is query-based pattern matching. Although pattern matching does not require the explicit creation of labelled data, formulating the queries often entails some form of qualitative analysis, e.g., grounded theory [41]. Alternatively, language models can be employed when labelled data is unavailable. In such a scenario, one has to rely exclusively on the language model’s pre-training, without further specific training for the task at hand. For example, a language model could be asked the following: “Is this requirements statement functional or non-functional? [*an individual requirements statement*]”.

If labelled data is available, the next factor to consider is the volume of such data, as captured by decision node (d) in Figure 2. When a considerable amount of labelled data is available, there are two options: applying feature-based learning or utilizing the labelled data to “fine-tune” a language model. Fine-tuning involves adapting a pre-trained model to a specific task through targeted training. In cases of limited labelled data, achieving task adaptation for a language model with minimal examples – a technique known as few-shot learning – is likely to yield better results.

It is important to note that there is no general rule as to what constitutes “considerable” or “limited” labelled data. Several parameters such as the quality of labels, the complexity of the relationships to be learned, the number of classes (in classification) and the range of values (in regression), the amount of noise in the data, the dimensionality of the feature space (in case of feature-based learning), and the desired level of accuracy to achieve can influence data needs. As such, experimentation on a case-by-case basis is crucial to determine whether the labelled data at hand should be regarded as considerable or limited. In our experience, and for the sake of offering ballpark figures, having fewer than 100 data points tends to constitute a limited amount. A considerable amount of data, on the other hand, is likely to materialize within the range of 500 to 5000 labelled data points.

We need to highlight three important aspects related to the process in Figure 2. First, no individual decision model can encompass the full spectrum of techniques employed in NLP4RE. Our model aims to offer a simplified representation of common scenarios, rather than imposing constraints or promoting a lack of flexibility in technique selection. Second, the process is likely to evolve in the future to stay in step with NLP4RE research. In particular, capitalizing on the interactive capabilities of large language models, there is potential to further elaborate the process by considering prompting strategies and providing additional guidelines. However, due to the scarcity of NLP4RE approaches built on large language models, we have to defer doing so to the future. Finally, when selecting enabling techniques for analysis, the cost and environmental impact must be considered. Most notably, the resource-intensive nature of large language models requires justification, especially when alternatives cannot be dismissed due to compelling reasons such as lack of accuracy.

2.3 Post-processing

Post-processing – the third step in the process of Figure 1 – aims to enhance the results of the Analysis step or to adapt these results for human analysts’ better understanding. Post-processing is not needed in all NLP4RE solutions and is thus an optional step.

To illustrate a simple scenario where post-processing is required, let us consider the task of requirements identification. For this task, one may apply the following heuristic as a post-processing step: if all but one sentence in a passage are categorized as requirements (during the Analysis step), that lone sentence should be reclassified from a non-requirement to a requirement. This adjustment will likely increase the accuracy of requirements identification [2].

For a more advanced example of post-processing, let us consider requirements completion based on predictions by a language model. In this context, the language model is likely to generate a non-negligible number of predictions that are not useful (false positives) alongside the useful ones. To reduce the incidence of unuseful predictions in the final results, the development of a post-processing filter becomes essential [31].

In its simplest form, post-processing can be light, e.g., in the case of the heuristic mentioned earlier for requirements identification. In more complex scenarios, like the one mentioned above where predictions need to be filtered, additional enabling techniques might be needed to carry out post-processing.

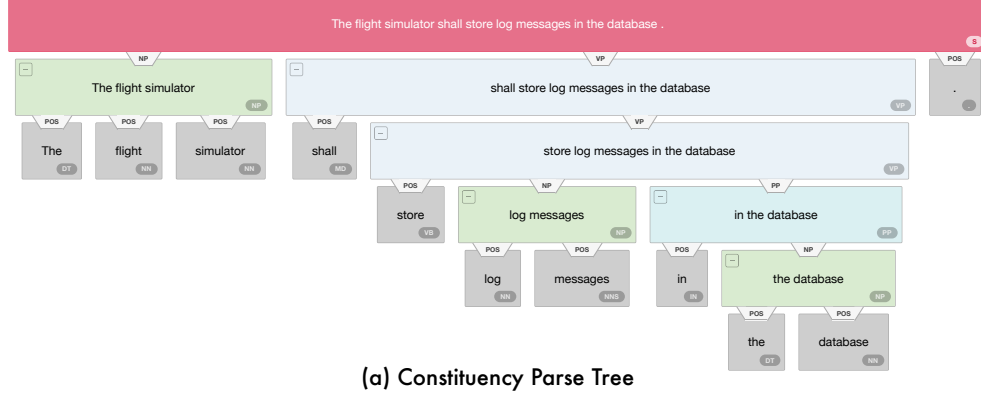
3 Enabling Techniques: Overview and Guidelines

In this section, we outline the various enabling techniques shown in Figure 1 and provide practical guidelines for applying and evaluating them.

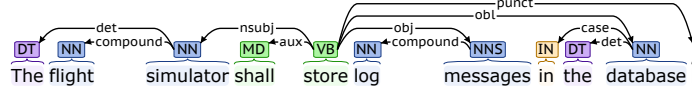
3.1 NLP Pipeline

The NLP pipeline is a sequence of modules that incrementally add linguistic annotations to an input text. The pipeline typically begins with tasks like tokenization (breaking text into words or sub-words), sentence splitting (segmenting a passage into sentences), and lemmatization (reducing words to their base forms). Next and depending on the annotations required, the pipeline performs tasks such as part-of-speech (POS) tagging (labelling words with their grammatical roles), named-entity recognition (identifying entities like names, dates, and locations), and syntactic parsing (analyzing sentence structure). Syntactic parsing includes two main techniques: constituency parsing (deconstructing sentences into grammatical constituents) and dependency parsing (determining grammatical relationships between words).

Figures 3 (a) and (b) respectively exemplify the annotations generated by the constituency parsing and dependency parsing modules for the following requirements sentence: $R =$ “*The flight simulator shall store log messages in the database.*”. The annotations generated by the constituency parser for this sentence are: NP (noun phrase), VP (verb phrase), and PP (prepositional phrase). These annotations capture the hierarchical constituents of the sentence. The relationships between the words in the sentence are given by the pairwise links generated through dependency parsing. For instance, from the dependency parse graph of Figure 3 (b), one can determine that “simulator” functions as the subject (nsubj) for “store,” and that “messages” serves as the object (obj) for this transitive verb. Crucially, both parsing methods require sentence splitting and POS tagging. Figure 3(a) illustrates sentence annotation (S),



(a) Constituency Parse Tree



(b) Dependency Parse Graph

Fig. 3 Illustration of (a) Constituency Parsing and (b) Dependency Parsing. Both Parsing Methods Require Sentence Detection and POS Tagging.

while both figures include POS tags (DT: determiner, NN: noun, NNS: plural noun, MD: modifier, VB: verb, IN: preposition).

When using the NLP pipeline, it is important to consider the specific natural language(s) that require support. While most NLP modules have good performance over well-written English, their effectiveness can vary significantly when dealing with other languages or when processing text that deviates from grammatical norms, such as user feedback (e.g., app reviews) or developer commit messages. Experimentation is therefore typically necessary for constructing an accurate NLP pipeline. NLP workbenches such as GATE [23] and DKPro [14] facilitate the integration of NLP modules from different NLP libraries. For instance, these workbenches enable the use of the POS tagger provided by one library, say, Apache OpenNLP [4], alongside the constituency and dependency parsers from another library, say, Stanford CoreNLP [32]. This flexibility to combine NLP modules from different libraries enables systematic experimentation with various pipeline configurations to determine the optimal configuration for the task at hand. An example of such experimentation can be found in our work on checking conformance with requirements templates [5].

If systematic experimentation with alternative NLP modules is not feasible or if there are constraints on using a single library, e.g., to minimize complexity, it would be important to conduct an error analysis on the annotations generated by the NLP pipeline. This analysis helps with ensuring the absence of systemic issues. Common

systemic issues include recurring errors in sentence detection, repeated inaccuracies in tokenization and POS tags, and incorrect parsing results.

💡 Takeaway: NLP Pipeline

Prioritize pipeline implementations that are known to work well for the language(s) you need to support. Experimentation with different NLP modules is often useful for improving accuracy. If extensive experimentation with the NLP pipeline is not possible or you are limited to a single library, conduct an error analysis on the annotations to identify and address any systemic issues such as recurring errors in sentence detection, POS tags, and parsing.

3.2 Relevance Measures

In the context of RE, *relevance* denotes the extent to which various requirement segments are related or how closely a specific requirements segment aligns with a particular query, topic, or artifact (e.g., a design document or a portion thereof). The concept of relevance is typically quantified using one or a combination of three distinct categories of metrics: *syntactic*, *semantic*, and *statistical*.

- **Syntactic Relevance** is the string-based or structural relatedness between two or more text segments. There are numerous syntactic relevance measures. For a fairly comprehensive list of syntactic measures, consult [24]. An example metric in the syntactic category is Levenshtein [24]. This metric calculates the minimum number of single-character edits (insertions, deletions, or substitutions) required to transform one string into another. Levenshtein distance is usually normalized (scaled from 0 to 1) by dividing it by the maximum possible edit distance between the two strings. For instance, the (normalized) Levenshtein distance between the phrases “software system” and “software systems” is 0.9375, indicating that the two strings have considerable lexical overlap.
- **Semantic Relevance** goes beyond string matching to measure similarity between word meanings. Semantic relevance is typically quantified using metrics like PATH within a semantic network such as WordNet [37]. WordNet captures various relationships, such as hypernymy and hyponymy, which denote an “is-a” connection, and meronymy and holonymy, representing a “part-of” association. To illustrate, consider the relationship between “vehicle” and “car”, where “vehicle” serves as the hypernym of “car”, and the relationship between “wheel” and “car”, with “wheel” acting as the meronym of “car”. PATH similarity calculates the shortest path between two concepts on the “is-a” hierarchy. For instance, the PATH score between the terms “cat” and “mammal” is higher than between “cat” and “vehicle”, indicating that “cat” is semantically more related to “mammal” than “vehicle”.
- **Statistical Relevance** assesses relevance by analyzing the frequency and distribution of terms within a document, often employing algorithms such as TF-IDF and BM25. These algorithms are frequently normalized on a scale from 0 to 1 [12]. Statistical methods typically operate at the level of term frequency and do not necessitate

a pre-constructed semantic network. For instance, TF-IDF evaluates a term’s importance within a document based on its frequency of occurrence in that document, normalized by its frequency across the entire corpus. BM25, which extends TF-IDF, takes into account further factors like term saturation and document length. To illustrate, consider a corpus of requirements documents. A term like “user authentication” might appear infrequently but could be highly significant. BM25 can rank a document that extensively discusses this term higher than a document that only mentions it in passing, thereby indicating its greater relevance in the context of security requirements. Statistical measures are more commonly employed in information retrieval (IR) problems within NLP4RE, such as querying requirements [3].

Applications and Other Considerations. Relevance measures serve two main use cases in NLP4RE. The first is to calculate similarity either within a set of requirements or between requirements and textual segments found in other development artifacts. The second use case is to assess the relative significance of terms in documents. Frequently, relevance measures are employed as features in both supervised and unsupervised learning.

Different relevance measures can be combined to provide a more holistic characterization of relevance. For example, when tasked with constructing a requirements glossary, the combination of syntactic and semantic measures can help identify a wider range of variations among related domain terms. To illustrate, if our objective is to cluster terms related to a flight simulation system, we anticipate that “flight coordinates”, “aircraft position”, and all concepts associated with “flight positioning” should land in the same cluster [7]. Simultaneously applying both syntactic and semantic measures facilitates the determination of these terms being highly similar.

In relation to syntactic measures, it is important to note that, because requirements frequently manifest variability in phrasing, vocabulary selection and syntactic structure, techniques like lemmatization and tokenization are often required prior to computing syntactic measures. This preprocessing helps mitigate variability and enhances the accuracy of comparisons [7].

Finally, and in relation to semantic measures, we note that these measures are quickly being replaced by more advanced techniques, notably embeddings. Embeddings not only capture semantic similarity but also contextual similarity, as we discuss in Section 3.3. Furthermore, while lexical resources like WordNet offer the advantage of establishing human-interpretable connections between words, techniques such as embeddings provide a more nuanced characterization of meaning, although they may not be entirely interpretable by humans. Consequently, when the primary goal is the application of similarity metrics, rather than explaining relationships, there is often limited justification for employing semantic measures like PATH in future research.

💡 Takeaway: Relevance Measures

Combining different relevance measures often results in more accurate analytical outcomes. Relevance measures can be applied at different levels of granularity, ranging from individual tokens (e.g., Levenshtein distance) to entire documents (e.g., TF-IDF). Empirical evaluations of relevance measures can centre around identifying the most effective combination of these measures or benchmarking advanced solutions against relevance measures considered as baseline methods.

3.3 Embeddings

Embeddings enable the conversion of words into numerical vectors. These vectors encapsulate the semantic connections among words, in turn supporting a more meaningful manipulation of language. Word embeddings are typically derived through self-supervised approaches such as Word2Vec [34], GloVe [36], and the pre-training of language models such as BERT [18] and GPT [38]. For example, using the 300-dimensional variant of GloVe embedding vectors, the word “requirements” would be represented as a 300-dimensional vector: $[-1.3598\text{e-}01, -1.8174\text{e-}01, \dots, -6.2015\text{e-}02]$.

While the primary goal of embeddings is to represent individual words, methods also exist for generating embeddings for phrases and sentences. For example, a simple approach for obtaining sentence embeddings is to compute the weighted average of the word embeddings in a given sentence [10].

There are two main use cases for embeddings in the existing NLP4RE literature: (1) computing semantic similarity, typically through the cosine measure, and (2) using embeddings as features for learning. To illustrate, suppose that we are interested in identifying most similar requirements, e.g., as a way to find overlapping or redundant requirements. Consider the following three statements: $R1 = \text{“The system shall react to user input within one second.”}$; $R2 = \text{“The system shall respond within one second.”}$; and $R3 = \text{“The system shall encrypt sensitive data.”}$. For a requirement sentence R , let $emb(R)$ denote the sentence’s embeddings. By utilizing the 300-dimensional variant of GloVe and employing averaging to derive sentence embeddings from word embeddings, we would obtain the following: $\cosine(emb(R1), emb(R2)) \approx 0.95 > \cosine(emb(R1), emb(R3)) \approx 0.83 > \cosine(emb(R2), emb(R3)) \approx 0.78$. Now, the requirements analyst can, for example, sort the requirements pairs in descending order of similarity and inspect the most similar pairs to determine if there are overlaps or redundancies. Alternatively, when it is feasible to create a labelled dataset for training, the embeddings can be used as features – either on their own or alongside other features – for building a feature-based classifier that distinguishes similar and non-similar requirements pairs. In our illustrative example, and assuming that only $R1$ and $R2$ are deemed similar, one could infer (among other labelled data points) the following for training: $emb(R1)|emb(R2)|SIMILAR$ and $emb(R1)|emb(R3)|NOT-SIMILAR$. Here, “|” denotes vector concatenation, and “SIMILAR” and “NOT-SIMILAR” denote labels for pairs of requirements.

There are some important considerations to note when working with embeddings:

Dimensionality of Embeddings. The dimensionality of embeddings, which refers to the number of dimensions (or features) used to represent each word as a vector, determines the richness of information in word vectors. While some dimensions may have clear interpretations, e.g., gender or sentiment, others can be complex, capturing subtle and abstract aspects of word semantics that would be difficult for humans to interpret directly. There is no universally suitable choice for the dimensionality of word embeddings. Although a larger dimensionality can provide increased semantic nuance and potentially improved accuracy, one must consider the curse of dimensionality [34] – the inherent challenges that arise when dealing with high-dimensional data. We recommend experimentation with alternative embedding methods and dimensionality options to find an acceptable trade-off between accuracy for the analytical task at hand and the challenges posed by high-dimensional data. Note that the techniques discussed in Section 3.5 for feature selection and reduction can also be applied to embeddings to reduce dimensionality.

Non-contextual vs. Contextual Embeddings. Embeddings can be either non-contextual or contextual. Non-contextual embeddings, such as those produced by GloVe, create fixed word vectors that remain the same regardless of context. In contrast, contextual word embeddings, as produced by models like BERT and GPT, consider the surrounding words to generate word vectors that adapt to the context. To illustrate, consider the following two sentences: $S = \text{“Meeting privacy requirements is essential.”}$; and $S' = \text{“The system shall meet all the privacy requirements stipulated by the GDPR.”}$ Using GloVe to obtain embeddings for the word “requirements” yields the same vector for both sentences. In contrast, if one uses models such as BERT or GPT, the embeddings obtained for the same word in different sentences would differ. For instance, the `bert-base-uncased` variant of BERT yields the following vectors for “requirements” in S and S' , respectively: $[2.7466\text{e-}01, 4.8193\text{e-}01, \dots, 4.1681\text{e-}01]$ and $[-1.6852\text{e-}01, 3.5143\text{e-}01, \dots, 3.5699\text{e-}01]$. This difference is due to the contextually different meaning of “requirements” in these two sentences, where the word conveys the sense of a prerequisite condition in S and the sense of a specification in S' .

Contextual embeddings offer greater accuracy but come at a higher computational cost. It is therefore important not to dismiss non-contextual embeddings outright but to weigh their potential alongside contextual embeddings to determine whether the added accuracy of contextual embeddings is worth the extra cost.

Domain-specific Embeddings. Domain-specific embeddings capture context-specific word meanings that generic embeddings may miss. For instance, specialized BERT variants, such as BioBERT [28] for biomedical texts and LegalBERT [15] for legal documents, provide domain-specific embeddings for their respective domains. When domain-specific embeddings exist, it is worthwhile to compare them with generic embeddings for potential improvements. Further, in cases where domain-specific embeddings are unavailable, but a suitable domain-specific corpus is accessible, one can

attempt to construct domain-specific embeddings from scratch. Recent efforts in software engineering, such as building Word2Vec and GloVe embeddings for model-driven engineering [30], provide useful guidance in this regard.

💡 Takeaway: Embeddings

Experiment with different embedding technologies and dimensionality options to strike a balance between accuracy and the challenges of high-dimensional data. Assess whether the added accuracy of contextual embeddings justifies their higher computational cost, or if non-contextual embeddings suffice for your specific task. Explore domain-specific embeddings when available, as they may be superior at capturing context-specific meanings.

3.4 Query-based Pattern Matching

The annotations produced by the NLP pipeline provide a rich basis for defining queries that detect patterns of interest within an input text. Typically, pattern-matching queries work with “spans”. Each span represents a distinct sequence of consecutive words or tokens within the given text.

An important technical aspect in pattern matching is the choice of the query language. In NLP, span information can be flat, focusing on individual words and their properties (like POS tags), hierarchical, revealing how words combine into larger structures (as in constituency parsing), or graph-based, capturing relationships between words (as in dependency parsing). Different NLP toolkits offer different query languages; CoreNLP [32], for instance, provides TokensRegex for token-based regular expressions, Tregex for tree-based linguistic structures, and Semgrex for syntactic dependency patterns. To illustrate, we exemplify these query languages:

- (a) The TokensRegex query `{tag:/VB.*/*}` extracts all spans tagged as verbs.
- (b) The Tregex expression `NP[<NN | <NNS]` extracts all spans that are Noun Phrases (NPs) and immediately dominate a singular noun (NN) or a plural noun (NNS).
- (c) The Semgrex query `{pos:/VB.*/*} >nsubj {} =subject >obj {} =object` extracts verbs that both have a subject and an object, alongside the subject and the object.

Figure 4 shows the results of the above queries as applied to the annotations in Fig. 3. All three queries serve a purpose in NLP4RE. Query (a) produces results that can be used as a feature for identifying requirements, grounded in the hypothesis that a higher verb count signifies a higher likelihood of requirements being present. Query (b) identifies constituent noun phrases and verb phrases within a requirements statement. This information is valuable for various purposes, one of which is to validate whether the statement conforms to a specific template, such as EARS [33]. Query (c) is useful for constructing a domain model [6]; the query extracts a probable association. In the case of our example, the association would be “[flight] simulator *stores* [log] messages”.

Query-based pattern matching techniques often face criticism related to both the scope of the study that informs query development and the applicability of the resulting queries beyond their initial purpose. To address these concerns, we recommend a

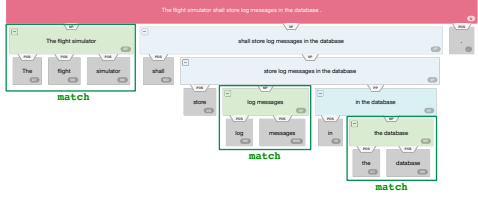
	Query	Matched Span(s)
(a)	TokensRegex: [{\tag:/VB.* /}]	<u>match</u> The flight simulator shall store log messages in the database .
(b)	Tregex: NP[<NN <NNS]	 The flight simulator shall store log messages in the database .
(c)	Semgrex: {pos:/VB.* /} >nsubj {}=subject >obj {}=object	<u>subject</u> <u>match</u> <u>object</u> The flight simulator shall store log messages in the database .

Fig. 4 Illustration of Query-based Pattern Matching over the Annotations of Fig. 3.

two-fold approach. First, it is important for query designers to carefully document the query specification process, providing a clear explanation of the considered documents and domains. The primary objective should be to enhance empirical reliability, ensuring that the process is as reproducible as possible by others. Second and concerning generalizability, one should avoid using all the text under analysis for query development. This approach allows for the assessment of query generalizability on unseen text, resulting in a more credible evaluation. An even more robust strategy involves expanding the analysis beyond the immediate domain under study. For example, if the focus is on regulatory documents, it is common to derive queries in one legal jurisdiction, such as Europe, and subsequently evaluate query accuracy in texts from another jurisdiction, like Canada [43]. By adhering to these basic principles — detailed documentation of the query derivation process and assessments of generalizability — one can enhance the utility of query-based pattern matching techniques.

💡 Takeaway: Query-based Pattern Matching

(1) When working with NLP annotations, select query language(s) that align with your specific analysis needs. (2) To ensure the reliability of query derivation, carefully document the steps you take to create the queries. This documentation should include the criteria, context, and content used for query design, making it easier for others to reproduce your work. (3) Refrain from using all the text you have for query development. To improve the applicability of your queries, evaluate their performance on unseen (or withheld) text or in different domains.

3.5 Feature-based Machine Learning

Feature-based machine learning (ML) is concerned with using labelled data to train algorithms to make predictions based on predefined features. Feature-based ML has numerous applications in NLP4RE. For instance, it can be used for (1) categorizing input requirements as either functional or non-functional [27], or (2) predicting the number of story points for user stories based on historical data [17]. The former application is a *classification* problem, entailing the assignment of discrete labels to data points (“functional” or “non-functional”); whereas the latter scenario constitutes a *regression* problem as it involves estimating numeric values (story points).

Building a feature-based ML model has five main steps. Below, we discuss these steps and outline some important practical considerations related to each step.

1. *Data collection* involves gathering relevant and sufficient data, including the labelling process. This phase can be very time- and effort-intensive, particularly when manual labelling is required. To ensure consistency, it is crucial to define a clear and agreed-upon labelling task with protocols that annotators can consistently follow. To enhance internal validity, authors should avoid annotating their test data. When feasible, it is further important to employ multiple annotators with some overlap to enable the calculation of inter-annotator agreement. Detailed documentation of the data-collection protocol is essential for reliability [2, 3].
2. *Data wrangling* is concerned with handling missing data, noise removal (e.g., stop words or redundant information), deciding about how to manage structured and unstructured data, and transforming results into the desired format. It is crucial to thoroughly document all techniques and decisions made in this step, as even minor changes can significantly impact the accuracy of the ML model [21].
3. *Feature engineering* is the process of defining relevant features for learning from the input data. In NLP4RE, features are usually defined over the outputs of the pre-processing techniques discussed in Sections 3.1–3.3. For instance, a feature based on the NLP pipeline could be the number of verbs in a sentence. A feature using relevance measures might be the highest TF-IDF rank of a specific term across all articles in a corpus. Finally and in relation to embeddings, the components within an embedding vector can be regarded as features, as elaborated in Section 3.3.

Features should be treated as hypotheses for addressing the problem at hand, and their usefulness must be evaluated. Techniques for computing feature importance, e.g., information gain [11], and PCA analysis [1], may be used for eliminating redundant or less significant features, thereby preventing overfitting. Feature engineering must also prioritize computational efficiency, particularly in real-time tasks such as live editing of requirements [2], where almost instant feature computation is necessary. In such scenarios, certain computationally expensive features may need to be sacrificed, even if they offer marginal improvements. For example, the resource-intensive process of constituent parsing, despite its potential value, may prove impractical. Conducting a cost-effectiveness analysis, weighing the cost of computing each feature against its impact on the resulting ML model’s accuracy is therefore recommended.

4. *Model Selection, Tuning, and Training* is concerned with choosing an appropriate ML algorithm, fine-tuning its hyperparameters, and training it over feature-engineered data [22]. Initially, the computed feature data should be split into three sets: the training set (typically 70% of all the data), the validation set (usually 10% of the data) for model selection and hyperparameter optimization, and the test set (typically 20% of the data) to evaluate the trained ML model [39]. During dataset partitioning, it is paramount to avoid data leakage, which means avoiding any exposure of test set data during training and validation [42]. If the dataset covers multiple projects, it is advisable to ensure that the test set includes data from ‘hold-out’ projects not involved in training or validation at all [39].

NLP4RE frequently encounters a scarcity of the kind of “abundant” data necessary for training complex ML algorithms, such as deep learning. In such instances, ML algorithms like Random Forest can provide scalable solutions, even when dealing with smaller datasets, noting of course that this advantage comes at the cost of requiring manual feature engineering.

Model tuning is another vital step in the process. While hyperparameter optimization should theoretically be integrated with model selection, the number of combinations to evaluate when combining model selection and hyperparameter optimization can be exceedingly large [22]. For example, systematically exploring discrete values across key hyperparameters in the Random Forest algorithm can result in over 500 combinations. Given this challenge, a more practical approach would be to initially select a suitable ML algorithm by experimenting with multiple algorithms at their default settings, and then proceed to fine-tune the hyperparameters for the chosen algorithm. The choice of hyperparameter tuning strategy depends on available resources, such as an extensive grid search or a lightweight stochastic random search, for example.

5. *Evaluation* involves assessing a trained ML model’s performance on a test set using metrics such as accuracy, precision, recall, F-score, and mean absolute error (MAE). Depending on the context, one can opt for “hold-out” test sets or k-fold cross-validation. The selection and prioritization of metrics require special attention. For instance, in a binary classification task where 95% of samples belong to Class A and only 5% to Class B, a classifier exclusively predicting Class A would achieve 95% accuracy. Consequently, accuracy may not be a suitable metric when the dataset is imbalanced. Another notable point related to metrics is that, in NLP4RE, recall often outweighs precision in terms of importance, typically making recall a priority in solution development [13]. In relation to reporting, care needs to be taken when reporting aggregate metrics like F-score: such metrics should be reported alongside the source metrics (in this case, precision and recall) rather than as substitutes.

Beyond reporting metrics, it is important to reflect on how these metrics translate into either benefits or drawbacks for users. For example, it is always valuable to contemplate the significance of various types of classification errors and determine their impact on users. When the cost of prediction errors is deemed too high, e.g., when the ML model’s predictions serve as recommendations requiring meticulous manual follow-up, one may consider implementing a human feedback loop to continuously retrain the model and reduce errors [8].

💡 Takeaway: Feature-based ML

Develop explicit procedures for manual data labelling during data collection and document these procedures. Keep track of data-wrangling decisions, as they can have a substantial impact on the outcomes. Approach features as hypotheses and confirm their significance before incorporating them into the final solution. Given the numerous combinations available for hyperparameter tuning during model selection and tuning, you may want to begin with experimentation using the default settings of machine learning algorithms. Ensure there is no data leakage, which means avoiding the inadvertent exposure of parts of test documents during the training and validation processes. The choice of evaluation metrics is of great importance and should be aligned with the specific NLP4RE task at hand.

3.6 Clustering Algorithms

Clustering algorithms group similar data points into subsets or clusters to reveal patterns and structures within the data. This is achieved using a quantitative measure of similarity and ensuring that points in the same cluster are more similar to each other than to those in different clusters. In NLP4RE, relevance measures (Section 3.2) and embeddings (Section 3.3) are commonly used to compute similarity for clustering purposes. For instance, using GloVe embeddings, requirements statements can be transformed into vectors and then clustered using clustering algorithms such as K-means, agglomerative clustering or expectation maximization (EM) [46]. To illustrate, consider a system with six requirements: R1-R3 from Section 3.3 and R4-R6 defined as follows: R4 = “The system shall allow users to customize the UI theme, available as ‘dark’ and ‘light’ versions.”; R5 = “The system shall allow users to sync data across multiple devices.”; and R6 = “The system shall allow users to reset passwords only after email authentication.”. Once sentence embeddings have been generated, cosine similarity between requirements pairs can be used as the basis for clustering. An illustrative clustering of these requirements, aimed at determining implementation responsibilities, might consist of [R1, R2] (system responsiveness), [R3, R6] (system protection), [R4] (system UI customization), and [R5] (system data management). Clustering has numerous applications in NLP4RE, including tasks such as traceability link retrieval, identifying overlapping or redundant requirements, and categorizing app reviews to pinpoint new feature requests. Below, we present some important practical considerations for an efficient utilization of clustering algorithms in NLP4RE.

Determining the Number of Clusters (k). Choosing an appropriate number of clusters (k) is an important prerequisite for effectively applying many common clustering algorithms such as K-means and EM. Techniques like the Elbow method, Silhouette analysis, and Bayesian Information Criterion (BIC) can help estimate an appropriate k [46]. Another alternative is a recent summarization metric proposed in clustering app reviews [35]. Domain knowledge remains a key factor alongside these methods in deciding the number of clusters. The appropriate value of k is influenced by the task

at hand and how users plan to utilize the resulting clusters. For instance, when clustering requirements terms, one may emphasize creating numerous small clusters, e.g., to simplify glossary construction [7]. On the other hand, when clustering core requirements within software product lines, it may be preferable to have a smaller number of clusters, as this streamlines the reviewing of common functions [40].

Hierarchical vs. Partitional vs. Soft Clustering. In partitional clustering algorithms, such as K-means, data is divided into non-overlapping clusters without any inherent structure. In contrast, hierarchical clustering creates a tree-like structure. Partitional methods are more suitable when the primary focus of the analysis is to rapidly identify overarching themes. For instance, Di Sorbo et al. [44] adopt a partitional approach to categorize app reviews into pre-defined topics, determining the necessary changes in the apps according to user feedback. Hierarchical clustering, on the other hand, is better suited when there is a need to comprehend and visualize the relationships and nested structures within the dataset. For instance, Reinhartz-Berger and Kemelman [40] use hierarchical clustering to explore the relationships between software product line requirements from different products by clustering the core requirements. It is worth noting that hierarchical clustering algorithms also provide mechanisms to create partitions. In soft clustering (also known as fuzzy clustering), each data point can belong to multiple clusters with varying degrees of membership. Soft clustering is ideal for situations where the boundaries between clusters are not rigidly defined, and instances may possess properties of multiple clusters. An example of this approach is the EM algorithm. An RE-related use case is clustering of requirements terms. In this context, it is beneficial to allow each term to have membership in different clusters [7]. For instance, the term “flight simulator” may find membership in both the “flight”-related and “simulator”-related clusters within a requirements document. Ultimately, the choice between partitional, hierarchical, or soft clustering depends on the specific task and analysis objectives at hand. Analysts should therefore understand the distinctions and potential applications of different types of clustering algorithms to ensure that their chosen algorithm aligns with the goals of their analysis.

Evaluation. Cluster evaluation can be either *internal* or *external*. Internal evaluation assesses the quality of clusters without relying on external labels, focusing on the principles of *cohesion* and *separation*. Cohesion measures how closely related the members of the same cluster are, reflecting the compactness of the clusters, while separation assesses how distinct or well-separated a cluster is from others. A higher separation implies that clustering can more effectively distinguish between clusters. External evaluation, on the other hand, measures the quality of clusters by comparing them to a ground truth. This comparison can involve evaluating the overlap or mutual information between the original clusters and the generated ones. Developing a ground truth in clustering is challenging due to the inherent subjectivity of this task and its context dependence. Requirements and related artifacts can have multifaceted interpretations, leading to multiple valid ways of clustering based on different perspectives or objectives. Creating a good ground truth necessitates: (i) a clear understanding of the concept of clustering as well as the end goal to achieve, (ii) substantial time and effort, and (iii) vigilance against bias and inconsistency, as different experts may have

contrasting opinions based on their experiences, potentially resulting in inconsistencies and posing threats to both internal and construct validity.

💡 Takeaway: Clustering Algorithms

Important decisions in clustering include determining a suitable number of clusters (e.g., the Elbow method or Silhouette analysis), and opting between clustering types (partitional, hierarchical, or soft). The choice of clustering algorithm depends on the goal of the analysis and requires domain understanding. Evaluating the effectiveness of clustering is critical, which can be approached internally by assessing the cohesion and separation of clusters or externally by comparing against a ground truth. An external evaluation often poses challenges, as creating a ground truth for clustering is subjective and often requires substantial effort.

3.7 Large Language Models

Language models (LMs) are statistical models that use neural networks to predict the next word in a sequence based on preceding words. For example, given the text “Paris is the capital of”, a language model may predict the next word as “France”. Language models increasingly serve as the foundation for various downstream NLP tasks such as text completion, translation, and summarization. Large language models (LLMs) like GPT-4 and BERT are scaled-up versions of the LM concept, with billions and sometimes even trillions of parameters. The descriptor “large” in LLM pertains to the model’s size in terms of the number of parameters. These parameters represent the neural network layers’ weights that the model acquires through training. Generally, larger models tend to perform better at comprehending context, drawing inferences and producing answers that resemble human responses.

Hyperparameters. LLMs have a range of hyperparameters that can be adjusted during both training and fine-tuning. The hyperparameter profiles can vary among different LLMs, and the extent to which individual LLMs allow end-users to modify these hyperparameters also varies. Some common LLM hyperparameters include: (i) *learning rate*, determining how quickly the model adapts its weights during training; (ii) *number of epochs*, referring to how many times the model will go through the entire training dataset; (iii) *batch size*, denoting the number of training examples utilized in one iteration; (iv) *sequence length*, referring to the number of tokens that the model reads in one go; (v) *dropout rate*, referring to the fraction of input units to drop during training to mitigate overfitting; and (vi) *temperature*, a parameter that modulates the probability distribution over the predicted words, making the outputs more focused (lower values) or more random (higher values).

Extensive experimentation across a broad range of LLM hyperparameter combinations currently presents challenges due to constraints in both time and budget. Nevertheless, it remains essential to select a practical subset of hyperparameter combinations that can be explored within the available resources. This approach enables

a more informed understanding of how hyperparameters influence the performance of an LLM in conducting a specific task.

Prompting. Prompting refers to providing a specific input or query to guide a generative AI model in producing the desired output, such as text or images [25]. Prompts, which are concise textual inputs given to generative AI models, including LLMs, convey information about the specific task that the model is expected to execute.

Creating prompts that are effective in eliciting the desired output from an LLM requires good prompt engineering. Prompt engineering involves the selection of appropriate prompt patterns and prompting techniques [45]. Prompt patterns encompass various templates tailored for specific objectives. For example, the output customization pattern focuses on refining the format or structure of LLM-generated output, with the LLM often adopting a particular persona (role) while generating the output. Prompting techniques, on the other hand, are strategies to extract the best possible output from LLMs. Example prompting techniques include zero-shot prompting, few-shot prompting, chain-of-thought prompting, and tree-of-thought prompting [29].

When conducting empirical examinations of LLM-based solutions, it is important to take into account the alternative choices that one can make during prompt engineering. These alternatives should ideally be compared through empirical means. Nonetheless, much like the challenges encountered when exploring hyperparameters, the vast array of possible combinations of prompt patterns and prompting strategies may be too numerous to thoroughly investigate. Another important aspect to consider is that even minor alterations in prompts can lead to considerable variation in the outputs generated by LLMs. In view of this, empirical studies should also assess prompt robustness by examining multiple variants of the same prompt.

Fine-tuning. While LLMs come pre-trained on very large corpora, they often require some level of fine-tuning to specialize them for particular tasks or domains. When the requirements automation task at hand aligns closely with common knowledge, such as ambiguity handling, one might achieve good results with little or no fine-tuning. However, if the task is specialized and involves RE-specific semantics, such as requirements classifications, fine-tuning often becomes necessary to ensure accurate results. When fine-tuning an LLM, one needs to be cognizant of the non-deterministic nature of the process, which is influenced by factors such as random initialization and regularization. Depending on the extent and diversity of the fine-tuning data, significant variations may be observed in results across different fine-tuning attempts. To account for this randomness, we recommend fine-tuning LLMs multiple times and reporting average results rather than relying on the outcome of a single fine-tuned model.

🔑 Takeaway: Large Language Models

When working with LLMs, explore hyperparameters within resource constraints. Invest in prompt engineering, using suitable prompt patterns and techniques, and empirically compare different alternatives. Take note of the non-deterministic nature of fine-tuning, and present results averaged over multiple fine-tuning runs.

4 Summary and Conclusion

The main goal of this chapter was to aid newcomers in the selection of appropriate NLP4RE techniques and the application of some essential principles for their evaluation. We acknowledge the wide array of NLP technologies employed in RE. Not all NLP4RE approaches may neatly align with the general, and sometimes simplified, framework we have outlined in this chapter.

This chapter was written during a transformative period in the NLP field, spurred by the emergence of generative AI and large language models. We hope that this chapter can serve as a stepping stone for quickly grasping the NLP technologies most relevant to RE. With both the NLP and RE landscapes constantly evolving, our hope is to maintain this chapter as a living document, continuously integrating emerging trends and pertinent techniques for automated requirements analysis.

Acknowledgements. The first author is grateful for the financial support provided by the Natural Sciences and Engineering Research Council of Canada (NSERC) through the Discovery and Discovery Accelerator programs.

References

- [1] Abdi H, Williams LJ (2010) Principal component analysis. Wiley interdisciplinary reviews: computational statistics 2(4):433–459. URL <https://api.semanticscholar.org/CorpusID:122379222>
- [2] Abualhaija S, Arora C, Sabetzadeh M, et al (2020) Automated demarcation of requirements in textual specifications: a machine learning-based approach. Empirical Software Engineering 25(6):5454–5497. <https://doi.org/10.1007/S10664-020-09864-1>
- [3] Abualhaija S, Arora C, Sleimi A, et al (2022) Automated question answering for improved understanding of compliance requirements: A multi-document study. In: 30th IEEE International Requirements Engineering Conference, RE, pp 39–50, <https://doi.org/10.1109/RE54965.2022.00011>
- [4] Apache OpenNLP (2006) Apache. URL <http://opennlp.apache.org>, last accessed: October 2023
- [5] Arora C, Sabetzadeh M, Briand L, et al (2015) Automated checking of conformance to requirements templates using natural language processing. IEEE Trans Software Eng 41(10):944–968. <https://doi.org/10.1109/tse.2015.2428709>
- [6] Arora C, Sabetzadeh M, Briand L, et al (2016) Extracting domain models from natural-language requirements: approach and industrial evaluation. In: Baudry B, Combemale B (eds) Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems, MODELS, pp 250–260, <https://doi.org/10.1145/2976767.2976769>
- [7] Arora C, Sabetzadeh M, Briand L, et al (2017) Automated extraction and clustering of requirements glossary terms. IEEE Trans Software Eng 43(10):918–945. <https://doi.org/10.1109/tse.2016.2635134>
- [8] Arora C, Sabetzadeh M, Briand LC (2019) An empirical study on the potential usefulness of domain models for completeness checking of requirements. Empir Softw Eng 24(4):2509–2539. <https://doi.org/10.1007/S10664-019-09693-X>

- [9] Arora C, Grundy J, Abdelrazek M (2023) Advancing requirements engineering through generative ai: Assessing the role of LLMs. CoRR abs/2310.13976. <https://doi.org/10.48550/ARXIV.2310.13976>
- [10] Arora S, Liang Y, Ma T (2017) A simple but tough-to-beat baseline for sentence embeddings. In: 5th International Conference on Learning Representations, ICLR, URL <https://openreview.net/forum?id=SyK00v5xx>
- [11] Azhagusundari B, Thanamani AS, et al (2013) Feature selection based on information gain. International Journal of Innovative Technology and Exploring Engineering 2(2):18–21. URL <https://api.semanticscholar.org/CorpusID:212611078>
- [12] Baeza-Yates RA, Ribeiro-Neto B (1999) Modern information retrieval. Addison-Wesley Longman Publishing Co., Inc., USA, <https://doi.org/10.5555/553876>
- [13] Berry DM (2021) Empirical evaluation of tools for hairy requirements engineering tasks. Empir Softw Eng 26(5):111. <https://doi.org/10.1007/S10664-021-09986-0>
- [14] de Castilho RE, Gurevych I (2014) A broad-coverage collection of portable NLP components for building shareable analysis pipelines. In: Ide N, Grivolla J (eds) Proceedings of the Workshop on Open Infrastructures and Analysis Frameworks for HLT, OIAF4HLT@COLING, pp 1–11, <https://doi.org/10.3115/V1/W14-5201>
- [15] Chalkidis I, Fergadiotis M, Malakasiotis P, et al (2020) LEGAL-BERT: the muppets straight out of law school. CoRR abs/2010.02559. URL <https://arxiv.org/abs/2010.02559>
- [16] Chen B, Chen K, Hassani S, et al (2023) On the use of GPT-4 for creating goal models: An exploratory study. In: Schneider K, Dalpiaz F, Horkoff J (eds) 31st IEEE International Requirements Engineering Conference, RE, Workshops, pp 262–271, <https://doi.org/10.1109/REW57809.2023.00052>
- [17] Choetkiertikul M, Dam HK, Tran T, et al (2019) A deep learning model for estimating story points. IEEE Trans Software Eng 45(7):637–656. <https://doi.org/10.1109/TSE.2018.2792473>
- [18] Devlin J, Chang M, Lee K, et al (2019) BERT: pre-training of deep bidirectional transformers for language understanding. In: Burstein J, Doran C, Solorio T (eds) Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT, pp 4171–4186, <https://doi.org/10.18653/V1/N19-1423>
- [19] Ezzini S, Abualhaija S, Arora C, et al (2022) Automated handling of anaphoric ambiguity in requirements: A multi-solution study. In: XXXX (ed) 44th IEEE/ACM 44th International Conference on Software Engineering, ICSE, pp 187–199, <https://doi.org/10.1145/3510003.3510157>
- [20] Ezzini S, Abualhaija S, Arora C, et al (2023) AI-based question answering assistance for analyzing natural-language requirements. In: XXXX (ed) 45th IEEE/ACM International Conference on Software Engineering, ICSE, pp 1277–1289, <https://doi.org/10.1109/ICSE48619.2023.00113>
- [21] Fan Y, Arora C, Treude C (2023) Stop words for processing software engineering documents: Do they matter? In: 2nd IEEE/ACM International Workshop on Natural Language-Based Software Engineering, NLBSE@ICSE. IEEE, pp 40–47, <https://doi.org/10.1109/NLBSE59153.2023.00016>

- [22] Feurer M, Hutter F (2019) Hyperparameter Optimization, Springer International Publishing, Cham, pp 3–33. https://doi.org/10.1007/978-3-030-05318-5_1
- [23] GATE NLP Workbench (2020) GATE. URL <http://gate.ac.uk/>, last accessed: October 2023
- [24] Gomaa WH, Fahmy AA (2013) A survey of text similarity approaches. International Journal of Computer Applications 68(13):13–18. <https://doi.org/10.5120/11638-7118>
- [25] Hariri W (2023) Unlocking the potential of chatgpt: A comprehensive exploration of its applications, advantages, limitations, and future directions in natural language processing. CoRR abs/2304.02017. <https://doi.org/10.48550/ARXIV.2304.02017>
- [26] Jain C, Anish PR, Singh A, et al (2023) A transformer-based approach for abstractive summarization of requirements from obligations in software engineering contracts. In: Schneider K, Dalpiaz F, Horkoff J (eds) Proceedings of the 31st IEEE International Requirements Engineering Conference, (RE’23), pp 169–179, <https://doi.org/10.1109/RE57278.2023.00025>
- [27] Kurtanovic Z, Maalej W (2017) Automatically classifying functional and non-functional requirements using supervised machine learning. In: Moreira A, Araújo J, Hayes J, et al (eds) 25th IEEE International Requirements Engineering Conference, RE, pp 490–495, <https://doi.org/10.1109/RE.2017.82>
- [28] Lee J, Yoon W, Kim S, et al (2020) Biobert: a pre-trained biomedical language representation model for biomedical text mining. Bioinform 36(4):1234–1240. <https://doi.org/10.1093/BIOINFORMATICS/BTZ682>
- [29] Liu P, Yuan W, Fu J, et al (2023) Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. ACM Comput Surv 55(9):195:1–195:35. <https://doi.org/10.1145/3560815>
- [30] López JAH, Durá C, Cuadrado JS (2023) Word embeddings for model-driven engineering. In: 2023 ACM/IEEE 26th International Conference on Model Driven Engineering Languages and Systems (MODELS), pp 151–161, <https://doi.org/10.1109/MODELS58315.2023.00036>
- [31] Luitel D, Hassani S, Sabetzadeh M (2023) Using language models for enhancing the completeness of natural-language requirements. In: Ferrari A, Penzenstadler B (eds) Requirements Engineering: Foundation for Software Quality - 29th International Working Conference, REFSQ, pp 87–104, https://doi.org/10.1007/978-3-031-29786-1_7
- [32] Manning CD, Surdeanu M, Bauer J, et al (2014) The stanford corenlp natural language processing toolkit. In: Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL, pp 55–60, <https://doi.org/10.3115/V1/P14-5010>
- [33] Mavin A (2012) Listen, then use EARS. IEEE Softw 29(2):17–18. <https://doi.org/10.1109/MS.2012.36>
- [34] Mikolov T, Chen K, Corrado G, et al (2013) Efficient estimation of word representations in vector space. In: Bengio Y, LeCun Y (eds) 1st International Conference on Learning Representations, ICLR, URL <http://arxiv.org/abs/1301.3781>

- [35] Nema P, Anthonysamy P, Taft N, et al (2022) Analyzing user perspectives on mobile app privacy at scale. In: 44th IEEE/ACM 44th International Conference on Software Engineering, ICSE, pp 112–124, <https://doi.org/10.1145/3510003.3510079>
- [36] Pennington J, Socher R, Manning CD (2014) Glove: Global vectors for word representation. In: Moschitti A, Pang B, Daelemans W (eds) Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP, pp 1532–1543, <https://doi.org/10.3115/V1/D14-1162>
- [37] Princeton University (2010) About WordNet. URL <https://wordnet.princeton.edu/>, last accessed: March 2019
- [38] Radford A, Narasimhan K, Salimans T, et al (2018) Improving language understanding by generative pre-training. OpenAI URL <https://api.semanticscholar.org/CorpusID:49313245>
- [39] Raschka S (2018) Model evaluation, model selection, and algorithm selection in machine learning. CoRR abs/1811.12808. URL <http://arxiv.org/abs/1811.12808>
- [40] Reinhartz-Berger I, Kemelman M (2020) Extracting core requirements for software product lines. *Requir Eng* 25(1):47–65. <https://doi.org/10.1007/S00766-018-0307-0>
- [41] Saldaña J (2015) The coding manual for qualitative researchers. Sage Publications, Thousand Oaks, CA, USA, URL <https://us.sagepub.com/en-us/nam/the-coding-manual-for-qualitative-researchers/book273583>
- [42] Shabtai A, Elovici Y, Rokach L (2012) A Survey of Data Leakage Detection and Prevention Solutions. Springer, <https://doi.org/10.1007/978-1-4614-2053-8>
- [43] Sleimi A, Sannier N, Sabetzadeh M, et al (2021) An automated framework for the extraction of semantic legal metadata from legal texts. *Empir Softw Eng* 26(3):43. <https://doi.org/10.1007/S10664-020-09933-5>
- [44] Sorbo AD, Panichella S, Alexandru CV, et al (2016) What would users change in my app? summarizing app reviews for recommending software changes. In: Zimmermann T, Cleland-Huang J, Su Z (eds) Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE, pp 499–510, <https://doi.org/10.1145/2950290.2950299>
- [45] White J, Fu Q, Hays S, et al (2023) A prompt pattern catalog to enhance prompt engineering with chatgpt. CoRR abs/2302.11382. <https://doi.org/10.48550/ARXIV.2302.11382>
- [46] Witten IH, Frank E, Hall MA (2011) Data mining: practical machine learning tools and techniques, 3rd Edition. Morgan Kaufmann, Elsevier, URL <https://www.worldcat.org/oclc/262433473>