Automated Handling of Anaphoric Ambiguity in Requirements: A Multi-solution Study

Saad Ezzini
University of Luxembourg
Luxembourg
saad.ezzini@uni.lu

Sallam Abualhaija
University of Luxembourg
Luxembourg
sallam.abualhaija@uni.lu

Chetan Arora
Deakin University
Australia
chetan.arora@deakin.edu.au

Mehrdad Sabetzadeh
University of Ottawa
Canada
m.sabetzadeh@uottawa.ca

ABSTRACT

Ambiguity is a pervasive issue in natural-language requirements. A common source of ambiguity in requirements is when a pronoun is anaphoric. In requirements engineering, anaphoric ambiguity occurs when a pronoun can plausibly refer to different entities and thus be interpreted differently by different readers. In this paper, we develop an accurate and practical automated approach for handling anaphoric ambiguity in requirements, addressing both ambiguity detection and anaphora interpretation. In view of the multiple competing natural language processing (NLP) and machine learning (ML) technologies that one can utilize, we simultaneously pursue six alternative solutions, empirically assessing each using a collection of ≈1,350 industrial requirements. The alternative solution strategies that we consider are natural choices induced by the existing technologies; these choices frequently arise in other automation tasks involving natural-language requirements. A side-by-side empirical examination of these choices helps develop insights about the usefulness of different state-of-the-art NLP and ML technologies for addressing requirements engineering problems. For the ambiguity detection task, we observe that supervised ML outperforms both a large-scale language model, SpanBERT (a variant of BERT), as well as a solution assembled from off-the-shelf NLP coreference resolvers. In contrast, for anaphora interpretation, SpanBERT yields the most accurate solution. In our evaluation, (1) the best solution for anaphoric ambiguity detection has an average precision of ≈60% and a recall of 100%, and (2) the best solution for anaphora interpretation (resolution) has an average success rate of ≈98%.

KEYWORDS

Requirements Engineering, Natural-language Requirements, Ambiguity, Natural Language Processing (NLP), Machine Learning (ML), Language Models, BERT.

ACM Reference Format:
Saad Ezzini, Sallam Abualhaija, Chetan Arora, and Mehrdad Sabetzadeh. 2022. Automated Handling of Anaphoric Ambiguity in Requirements: A Multi-solution Study. In *44th International Conference on Software Engineering (ICSE '22), May 21–29, 2022, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3510003.3510157

ICSE '22, May 21–29, 2022, Pittsburgh, PA, USA

ACM ISBN 978-1-4503-9221-1/22/05.
https://doi.org/10.1145/3510003.3510157

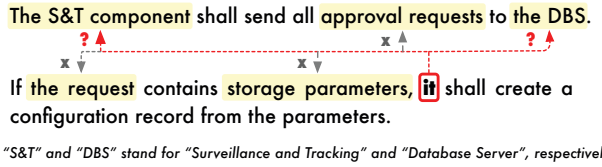
1 INTRODUCTION

Natural language (NL) is the most common medium for specifying systems and software requirements. NL enables communication between stakeholders who may have different backgrounds, often requiring little or no additional training [71]. NL requirements are nonetheless prone to defects such as ambiguity [10, 20, 71]. Ambiguity occurs when a word, phrase or sentence is open to multiple interpretations [69]. Ambiguity can have a negative impact on the quality of requirements and also potentially jeopardize the success of a project [24, 41]. A common cause of ambiguity in requirements is anaphora [28, 39, 81, 95].

Anaphora means repetition in Greek and is defined as references to entities mentioned earlier in the text. These references are called *anaphors* and the entities to which they refer are called *antecedents* [60]. Anaphoric ambiguity occurs when there is more than one plausible antecedent [24, 61]. In linguistics, there are several types of anaphora [60]. In requirements engineering (RE), anaphora is typically scoped to pronominal anaphora, i.e., when the anaphor is a *pronoun* [25, 95]. This is because pronominal anaphora has been clearly established as a genuine source of ambiguity in requirements [39]. *Anaphoric ambiguity detection* in RE is thus the task of identifying ambiguous occurrences of pronouns [94]. The closely related task of *anaphora resolution (interpretation)* is concerned with finding the most likely antecedent for a given pronoun [61].

To illustrate, consider the example in Figure 1. Here, the anaphor is *it*, occurring in the second sentence. The potential antecedents are the preceding noun phrases (NPs), namely "the S&T component", "approval requests", "the DBS", "the request" and "storage parameters". The pronoun *it* is unlikely to refer to "approval requests" or "storage parameters" due to number disagreement (here, singular pronoun versus plural NPs). Similarly, *it* is unlikely to refer to "the request", since *it* is the subject of the verb "create", and "the request" is not a suitable replacement for the subject of this verb. It is not entirely clear though whether *it* refers to "the S&T component" or "the DBS". Depending on which antecedent – "the S&T component" or "the DBS" – is selected, there are two different interpretations as to which subsystem should create a configuration record. To properly deal with this situation, the pronoun *it* has to be

either detected as *ambiguous* or resolved as referring to the *correct* antecedent, which happens to be "the S&T component". We note that identifying the correct antecedent in this example would likely be impossible without domain knowledge.



"S&T" and "DBS" stand for "Surveillance and Tracking" and "Database Server", respectively.

Figure 1: Example of Anaphoric Ambiguity.

In RE, unconscious disambiguation of requirements is common, particularly by stakeholders who have domain knowledge or a good understanding of the system under development [78]. Not all stakeholders in a project, however, share the same level of domain knowledge or familiarity with the system-to-be. For example, it is known that developers are prone to interpreting requirements inaccurately [63]. To reduce the potential for misinterpretation and the ensuing consequences, it is desirable to deal with ambiguity in requirements as early as possible. Typically, and somewhat in contrast to the literature on Natural Language Processing (NLP), RE prioritizes ambiguity detection over anaphora resolution. The rationale is that any genuine ambiguity in requirements needs to be inspected by human analysts and mitigated by rephrasing or other means, as opposed to the ambiguity being subjected to automated interpretation, i.e., what is commonly done in NLP [25].

Anaphoric ambiguity is prevalent in NL requirements. Estimates from the RE literature suggest that nearly 20% of industrial requirements contain anaphora [25, 81]. Current RE research on anaphoric ambiguity [7, 22, 25, 94, 95], as we elaborate in Section 2.2, does not adequately explore two important facets. First, the existing work relies primarily on the traditional methods in NLP and machine learning (ML). With the rapid emergence and adoption of new technologies such as pre-trained language models, BERT [18] being a notable example, the landscape for the processing (and generation) of NL content has changed drastically. This, on the one hand, provides an opportunity to develop new solutions, and, on the other hand, necessitates a revamp and reexamination of the existing solutions, now using better enabling technologies. Second, the existing RE solutions for anaphoric ambiguity have been evaluated on either a single application domain (e.g., railway) or on very small datasets. As such, empirical results remain scarce on the usefulness of automated techniques for dealing with anaphora in requirements documents.

On the surface, it may seem that one can readily adopt existing solutions from the NLP community to deal with anaphoric ambiguity in requirements. In the NLP literature, anaphora is typically addressed as part of *coreference resolution*, which is concerned with finding mentions that refer to the same entity in a given text [38, 76]. Coreference resolution is often an intermediate step for more advanced NLP tasks such as question answering and sentiment analysis [61]. As noted earlier, in RE, we prioritize detection over resolution, since we want to bring ambiguous cases to the analysts' attention for further examination. Existing coreference resolvers have not been built to support ambiguity detection, thus complicating the application of an individual resolver for this task.

Our aim in this paper is to arrive at a practical and effective solution for handling anaphoric ambiguity in textual requirements. By "handling" anaphoric ambiguity, we mean the primary task of detecting genuine cases of anaphoric ambiguity and the secondary task of interpreting (resolving) anaphora when the risk of ambiguity is sufficiently low. We achieve our aim by empirically investigating *multiple* solution strategies. Some of the investigated strategies are new and some are adaptations of existing work that are implemented using state-of-the-art technologies. The alternative strategies considered are choices that, in our experience, recurrently arise when engineering requirements automation solutions using NLP and ML. These choices particularly include: (1) whether to use hand-crafted language features, word embeddings or a combination thereof for classification, (2) whether pre-trained language models like BERT are a viable replacement for the more traditional techniques, and (3) whether a mashup of existing (and often generic) NLP tools would be adequate for specific RE tasks.

Our decision to examine and report on multiple solution strategies is motivated by building empirical insights about the mentioned choices. Naturally, our findings in this paper are limited to the task at hand, i.e., handling anaphoric ambiguity. Nonetheless, we believe that our mode of investigation contributes to establishing a framework for comparing the choices available in other requirements automation tasks that are addressed via NLP and/or ML.

Contributions. This paper makes the following contributions:

(1) We develop six alternative solutions for automated handling of anaphoric ambiguity in requirements. The solutions span both traditional as well as more recently established NLP and ML technologies. We implement all six solutions using Jupyter Notebooks [42], and make the solutions publicly available[1].

(2) We empirically evaluate the above-mentioned alternatives on two industrial datasets. The first dataset is a pre-existing one [2], containing 98 requirements with 109 pronoun occurrences. The second dataset was curated as part of our work using third-party (non-author) annotators. This second dataset is a collection of 22 industrial requirements specifications from eight different application domains and containing a total of 1,251 requirements with 737 pronoun occurrences. Over these datasets, for detecting anaphoric ambiguity, supervised ML classification yields the best results with an average precision of ≈60% and a recall of 100%. As for anaphora resolution, a fine-tuned language model from the BERT family of models turns out to be the best solution with a success rate of ≈98%. The fact that different best solutions emerge for two closely related tasks further signifies the usefulness of running multi-solution studies like ours.

Significance. The significance of our work is two-fold: (1) ambiguity handling is a major concern in RE. We devise an accurate automated solution to address a prevalent (and problematic) ambiguity type, namely anaphoric ambiguity; (2) the NLP landscape has evolved drastically in recent years. Comparing the more traditional techniques against new advancements is beneficial and relevant to many AI-based RE automation tasks beyond ambiguity handling.

[1] https://tinyurl.com/mww2w46t

We demonstrate how such comparisons can be made systematically. We further provide insights and lessons learned, and shed light on potential challenges.

Structure. Section 2 discusses background and positions our work against the related literature in NLP and RE. Section 3 presents our alternative solutions for handling anaphoric ambiguity in requirements. Section 4 reports on our empirical evaluation. Section 5 addresses threats to validity. Section 6 concludes the paper.

2 BACKGROUND AND RELATED WORK

This section presents the necessary background for our solutions and further discusses the related literature in RE and NLP.

2.1 Background

Below, we discuss the enabling technologies used by our solutions marked ① to ⑥ in Figure 3. The precise design of these alternative solutions will be elaborated in Section 3.

Language Models (LMs). LMs are statistical models that assign probabilities to words or phrases given some training corpus. For example, an LM would assign a higher probability to the phrase "briefed reporters on" than the phrase "briefed to reporters" [37]. BERT, standing for Bidirectional Encoder Representations from Transformers, is a pre-trained masked language model (MLM) introduced by Devlin et al. [18]. MLMs randomly mask a fraction of the tokens in the pre-training text (the BooksCorpus and English Wikipedia in the case of BERT); the pre-training objective is then to predict the original vocabulary of these masked tokens based on the surrounding context. For example, BERT should predict the masked token "briefed" in the phrase "`[MASK]` reporters on".

Pre-trained LMs can be employed to directly solve downstream NLP tasks such as anaphoric ambiguity handling (the focus of our work). We integrate LMs into our solutions using two strategies. The first strategy is to *fine-tune* the parameters of a pre-trained LM on a labeled dataset for anaphoric ambiguity handling. We apply this strategy to devise solutions ① and ② based on SpanBERT [36], a variant of BERT. In contrast to BERT, SpanBERT is pre-trained to predict masked text spans (rather than masked tokens). SpanBERT is better suited than BERT for tasks such as anaphora resolution and question answering where the output is a text span, e.g., a noun phrase rather than an individual noun [44]. The second strategy is to *extract contextual embeddings* from the pre-trained LM and use these embeddings as learning features in ML-based text classification. Embeddings are mathematical representations capturing the syntactic and semantic characteristics of text. For developing solution ④, we use embeddings from both BERT [18] and SBERT [77]. While BERT derives embeddings for individual tokens, SBERT is optimized for deriving semantically meaningful embeddings for an entire text sequence.

Machine Learning (ML). Supervised ML (including text classification [87]) requires labeled data consisting of datapoints described as a set of features and a class label. Using this labeled data, an ML classifier is trained to discriminate among the different classes based on the features. Subsequently, the classifier will be able to predict the class of a previously unseen datapoint described by the features. For text classification, different types of learning features can be used [3]. Among them, we apply in our work both manually-crafted features collected from the literature as well as contextual embeddings, presented earlier.

Solutions ③ and ④ – and also ⑤ which is a combination of ③ and ④ – are ML-based. In our labeled data, each datapoint is the combination of a pronoun and a candidate antecedent, both occurring in some context. Each datapoint is labeled *correct*, *incorrect* or *inconclusive*, as we explain in Section 3. Our empirical evaluation examines several widely used ML classification algorithms, namely decision tree (DT), feed-forward neural network (FNN), k-nearest neighbour (kNN), logistic regression (LR), naïve Bayes (NB), random forest (RF) and support vector machine (SVM). We refer the reader to textbooks for more details about these algorithms [30, 91].

Natural Language Processing (NLP) Pipeline. In our work, we apply an NLP pipeline composed of eight modules: (1) *tokenizer* for splitting the text into tokens; (2) *sentence splitter* for breaking up the text into individual sentences; (3) *part-of-speech (POS) tagger* for assigning a POS tag, e.g., noun, verb or pronoun, to each token in each sentence; (4) *lemmatizer* for identifying the canonical form (lemma) of each token, e.g., the lemma for "playing" is "play"; (5) *constituency parser* for identifying the structural units of sentences, e.g., NPs; (6) *dependency parser* for defining the grammatical dependencies between the tokens in sentences; (7) *coreference resolver* for finding mentions that refer to the same textual entity; and finally, (8) *semantic parser* for extracting information about words' meanings. Modules 1 to 6 are prerequisites for all our solutions (see the preprocessing step in Section 3.2). Our ML-based solutions additionally use modules 7 and 8 for extracting language features. Module 7 is the basis for solution ⑥.

2.2 Related Work

Ambiguity in natural language has been studied extensively [28, 54, 61]. In RE, different dimensions of ambiguity have been explored, including understanding the significance of ambiguity in requirements [24, 28, 32, 78, 84], analyzing the linguistic causes of ambiguity [10, 21, 39, 54], ambiguity prevention [4, 5, 55, 58, 80], and ambiguity detection and resolution [5, 17, 20, 23, 25, 29, 40, 43, 64, 81, 83, 88, 90, 95]. Below, we discuss related work on anaphoric ambiguity detection and anaphora resolution, covering both the RE and NLP communities.

In RE, anaphoric ambiguity has been addressed only to a limited extent, despite (pronominal) anaphora being a common source of misunderstandings in requirements [28]. Yang et al. [94, 95] propose an ML-based solution over language features for detecting cases of anaphoric ambiguity leading to misunderstandings. Using 200 anaphoric pronouns from different domains, they report an accuracy of ≈76% for classifying whether an antecedent is correct for a given pronoun. Detecting potential anaphoric ambiguity has also been addressed as a sub-topic of defects detection, with some basic solutions having been proposed, e.g., generating potential ambiguity warnings for all pronouns or only for pronouns whose surrounding text matches some simple syntactic patterns [5, 28, 81].

The approaches outlined above have two limitations. First, they are based on traditional technologies from NLP and ML – two fields that have advanced significantly over the past few years. Second, these approaches have been evaluated on small datasets or single

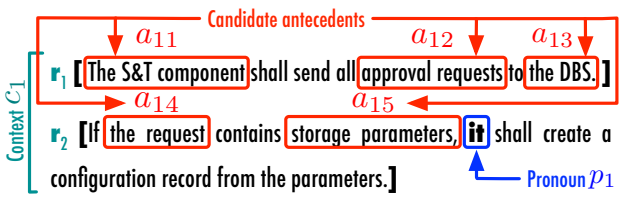


Figure 2: Illustration of our Notation.

domains. We address the first limitation by (i) devising solutions in view of recent advances in NLP and ML, particularly the emergence of pre-trained language models; and (ii) re-examining the state-of-the-art approach by Yang et al. [94, 95], enhanced with several new language features gleaned from the literature [13, 19, 47, 56, 62, 75]. To address the second limitation, we conduct a multi-solution empirical study including a relatively large RE dataset that covers eight different application domains.

In the NLP community, dealing with anaphora is a long-standing problem [61]. As already noted in the introduction section, compared to RE, the focus in NLP is primarily on anaphora resolution, given the needs of the NLP tasks that are further downstream [47, 70]. Despite numerous attempts at addressing anaphora resolution, the complex nature of the task has slowed progress for several anaphora types [86]. The anaphora resolution techniques in the NLP community are broadly classified into three categories: syntactic, semantic and neural-network-based [46]. The syntactic and semantic approaches focus on designing ML features based on grammatical structure and word meanings in sentences. In the neural-network-based approaches, anaphora resolution is often reformulated as a question-answering problem. Recent solutions in this category achieve promising results [33, 93].

In addition to being focused on resolution, the techniques developed by the NLP community are trained on generic corpora, e.g., Wikipedia. Due to the major differences between the terminology and style applied in requirements writing versus what is available in generic corpora [26], NLP tools usually do not work well if applied as-is to requirements documents [20, 90]. To address this problem, we collect and annotate, as part of our work, a dataset of industrial requirements. Taking inspiration from the state-of-the-art NLP directions, we build multiple solutions for handling anaphoric ambiguity, while ensuring that anaphoric ambiguity detection is explicitly addressed and prioritized over anaphora resolution.

3 SOLUTIONS DESIGN

We start this section by defining in an analytical manner anaphoric ambiguity detection and anaphora resolution. This is followed by a discussion of the preprocessing required for automating these tasks. We then present the design of six alternative solutions for automated handling of anaphoric ambiguity in requirements; these solutions will be tuned and evaluated in Section 4.

3.1 Problem Definition

Let $\mathcal{R} = (r_1, r_2, \ldots, r_n)$ be a sequence of requirements, where each r_i represents a single requirements sentence. Let $\mathcal{P} = (p_1, p_2, \ldots, p_m)$ be all the pronouns in $\mathcal{R}$ in their order of appearance. Following best practice [95], we define the context c_j of a pronoun p_j as two consecutive sentences $c_j = (r_{i-1}, r_i)$; $2 \leq i \leq n$, $1 \leq j \leq m$, where r_i is the sentence in which p_j occurs. If p_j occurs in r_1, then the context is one sentence only, i.e., $c_j = (-, r_1)$. Each pronoun occurrence is represented by a distinct $p_j \in \mathcal{P}$. This means that multiple occurrences of the same pronoun constitute different elements in $\mathcal{P}$, even when the occurrences are within the same sentence. For each $p_j \in \mathcal{P}$, the context of p_j, i.e., c_j, induces a set of candidate antecedents denoted $\mathcal{A}_j = \{a_{j1}, a_{j2}, \cdots, a_{jt}\}$.

To illustrate our notation, we recall the example of Figure 1. Let r_1 and r_2 be the two consecutive sentences in that example. Then, $\mathcal{R} = (r_1, r_2)$. There is only one pronoun in $\mathcal{R}$; therefore, $\mathcal{P} = (p_1)$ where $p_1 = it$. The context for p_1 is $c_1 = (r_1, r_2)$, and the set of candidate antecedents for p_1 is $\mathcal{A}_1 = \{a_{11}, a_{12}, a_{13}, a_{14}, a_{15}\}$ where a_{11} = *"the S&T component"*, a_{12} = *"approval requests"*, a_{13} = *"the DBS"*, a_{14} = *"the request"* and a_{15} = *"storage parameters"*. For easier referral later in the paper, we visually show in Figure 2 how our notation is applied to the example of Figure 1.

Using our notation, *anaphoric ambiguity detection* is to decide whether a given pronoun occurrence p_j is *ambiguous* or *unambiguous* in its context c_j. *Anaphora resolution* is to identify the most likely antecedent for p_j.

3.2 Preprocessing

The preprocessing step generates the input for the alternative ambiguity handling solutions that we consider in this paper. We first apply the NLP pipeline, discussed in Section 2.1, on a given requirements specification (RS) to parse its textual content. We create the list of all pronouns (i.e., $\mathcal{P}$) occurring in RS; this is done by selecting the words that the POS tagger marks as *PRP* (personal pronoun) or *PRP$* (possessive pronoun) [52]. For each $p_j \in \mathcal{P}$, we identify the context c_j as the requirement r_i in which p_j occurs and the preceding requirement r_{i-1} (for $i \geq 2$). Finally, for each p_j, we generate the set of all candidate antecedents $\mathcal{A}_j$. Since antecedents are NPs, as noted in Section 1, we generate $\mathcal{A}_j$ by including all NPs that precede p_j in c_j, as automatically identified by the constituency parser module in the NLP pipeline. We further include any segment following the pattern *[NP and/or NP]* (e.g., "the sender and the receiver") and *[NP preposition NP]* (e.g., "the component of the system"). Doing so improves the set of candidate antecedents by covering the cases where p_j refers to a compound NP [6, 95].

3.3 Alternative Solutions

We consider six alternative solutions for handling anaphoric ambiguity. These are shown in Figure 3. Alternatives ① and ② are based on SpanBERT; alternatives ③, ④ and ⑤ are based on supervised ML; and, alternative ⑥ is based on existing NLP coreference resolvers. We note that the expected input differs across solutions: The solutions based on SpanBERT take as input tuples of the form $\langle c_j, p_j \rangle$; the ML-based solutions take as input triples of the form $\langle c_j, p_j, a_{jk} \rangle$; and, the NLP-based solution take as input merely the context information (c_j) for pronoun occurrences. The input for all solutions is directly constructible from the preprocessing results.

Table 1 outlines for each solution the inputs, the intermediate outputs and the rules for processing the intermediate outputs. The

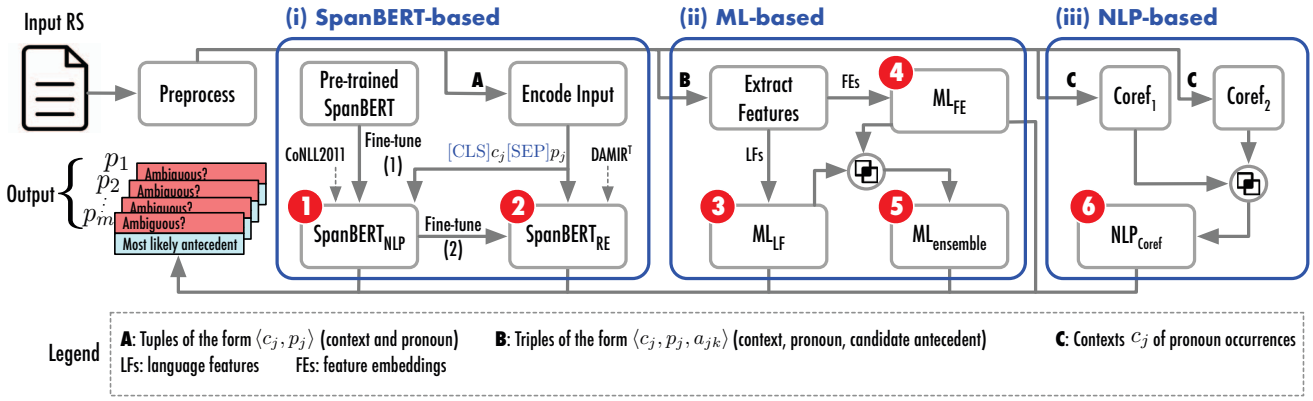


Figure 3: Overview of Solution Alternatives (marked ① to ⑥).

Table 1: Inputs, Intermediate Outputs and Ambiguity-handling Rules for Solution Alternatives.

Alternative(s)	Input (**I**), Intermediate Output (**O**) and Ambiguity Handling Rules (**R**)
① ②	**I**: $\langle c_j, p_j \rangle$ tuples. **O**: Tuples of the form $\langle s_q, pr_q \rangle$, where s_q is a text span and pr_q is the probability of s_q being the antecedent for p_j. **R**: (*Anaphora Resolution**) For a pronoun p_j, if there is exactly one s_q in c_j such that pr_q is $\geq$ a fixed (empirically tuned) threshold, then s_q is the most likely antecedent of p_j. *(Ambiguity Detection)* If such s_q is identified, then p_j is *unambiguous*; otherwise p_j is *ambiguous*.
③ ④ ⑤	**I**: $\langle c_j, p_j, a_{jk} \rangle$ triples. **O**: Tuples of the form $\langle \ell_{jk}, pr_{jk} \rangle$, where ℓ_{jk} is a label characterizing the referential relation between a_{jk} and p_j in c_j and where pr_{jk} is the prediction probability for ℓ_{jk}. For anaphora resolution, the labels admitted by ℓ_{jk} are *correct* and *incorrect*; for ambiguity detection, ℓ_{jk} additionally admits *inconclusive*. **R**: (*Anaphora Resolution**) For a given p_j, if there is exactly one a_{jx} such that ℓ_{jx} = *correct* with any probability, then a_{jx} is the most likely antecedent of p_j. Otherwise, if multiple ℓ_{jk} are predicted as *correct* for p_j, then we deem p_j's most likely antecedent to be a_{jx} where x is the index at which ℓ_{jx} has the highest probability pr_{jx}. *(Ambiguity Detection)* For a given p_j, if there is exactly one label ℓ_{jx} = *correct*, then p_j is *unambiguous* if, additionally, either of the following conditions hold: (a) pr_{jx} is $\geq$ a fixed (empirically tuned) threshold, or (b) there is no label ℓ_{jk} that is *inconclusive*. Otherwise, p_j is *ambiguous*.
⑥	**I**: Contexts (c_j) of pronoun occurrences. **O**: Each pronoun occurrence p_j alongside mentions m_1 and m_2 found by Coref$_1$ and Coref$_2$, respectively. **R**: (*Anaphora Resolution**) If $m_1 = m_2$, then m_1 (= m_2) is the most likely antecedent of p_j. *(Ambiguity Detection)* If an antecedent is identified by the anaphora resolution rule, then p_j is *unambiguous*; otherwise, p_j is *ambiguous*.

* *If no anaphora resolution rule is triggered for a given pronoun occurrence, then no antecedent is predicted.*

rules produce the final results for anaphora resolution and ambiguity detection. We elaborate our alternative solutions next.

(i) Solutions based on SpanBERT. We employ the recent language model SpanBERT [36], introduced in Section 2.1, to develop solutions ① and ②, referred to as SpanBERT$_{NLP}$ and SpanBERT$_{RE}$, respectively. We first fine-tune the pre-trained SpanBERT model to generate SpanBERT$_{NLP}$ using the *CoNLL2011* dataset [34, 53, 72] – a large dataset of generic text with about 7,000 pronoun occurrences. This fine-tuning step – fine-tune (1) in Figure 3 – aims to adjust the parameters of the general SpanBERT model using the inputs and outputs of *CoNLL2011* on the anaphora resolution task. Next, we fine-tune SpanBERT$_{NLP}$ to generate SpanBERT$_{RE}$ on a subset of *DAMIR* – a dataset of NL requirements, which we have constructed as part of our work. The second fine-tuning – fine-tune (2) in Figure 3 – enhances SpanBERT$_{NLP}$ by exposing it to examples of ambiguous and unambiguous pronouns from the RE domain. The hypothesis we would like to examine using the resulting solution, i.e., SpanBERT$_{RE}$, is whether requirements-specific knowledge improves the accuracy of anaphoric ambiguity handling in RE. The *CoNLL2011* and *DAMIR* datasets are discussed in Section 4.3.

The input to BERT and its variants needs to be tokenized and encoded into the same format used by the pre-trained models. Specifically, we encode each tuple $\langle c_j, p_j \rangle$ as [*CLS*]c_j[*SEP*]p_j. Two special tokens are automatically added by BERT's tokenizer: *[CLS]* to represent the classification output and *[SEP]* to separate c_j from p_j. Any repeated occurrence of the same pronoun p_j is replaced with p_j#d, where $d \geq 1$ is a unique identifier. The multiple occurrences are then encoded as [*CLS*]c_j[*SEP*]p_j#d. The [CLS] token is relevant for SpanBERT's pre-training, which is not part of our analysis. [CLS] thus has no significance for our analytical purposes.

The SpanBERT-based solutions, ① and ②, handle ambiguity using the respective rules provided in Table 1. There is a threshold

θ_α for controlling the resolution results. We recommend $\theta_\alpha = 0.9$ based on our tuning, discussed in Section 4.5. For the example in Figure 2, the input to ① and ② is $\langle c_1, p_1 \rangle$, encoded as $[CLS]c_1[SEP]p_1$. The intermediate output of both solutions would be a tuple like $\langle s_1 =$"the S&T component", $pr_1 = 0.99\rangle$. The text span s_1 would be the antecedent of p_1, since it is identified with a probability ≥ 0.9. Thus, p_1 would be detected as unambiguous. Note that ① and ② do no necessarily demarcate all possible text spans in c_j, but rather only those that the solutions find relevant for anaphora resolution.

(ii) Solutions based on supervised ML. We refer to our three ML-based solutions, ③, ④ and ⑤, as ML_{LF}, ML_{FE} and $\text{ML}_{ensemble}$, respectively. ML_{LF} is trained on 45 *language features (LFs)* collated from the existing NLP and RE literature on anaphora resolution [13, 19, 47, 56, 62, 75, 95]. The description of the LFs is provided online [82]. ML_{FE} is trained on *feature embeddings (FEs)* which are contextual representations of the input, as explained in Section 2.1. $\text{ML}_{ensemble}$ is an ensemble classifier which combines the results of ML_{LF} and ML_{FE}.

Each triple $\langle c_j, p_j, a_{jk} \rangle$ in the input to the ML-based solutions needs to be transformed into a feature vector. ML_{LF} is built over 45-dimensional feature vectors encoding the LFs. The values for the LFs are computed using the NLP pipeline. The LFs characterize the referential relation between p_j and its candidate antecedent a_{jk}, e.g., number agreement when both are plurals or singulars. ML_{FE} is built over 768-dimensional feature vectors representing the FEs extracted from BERT [18] and SBERT [77]. The FEs capture the semantic and syntactic regularities of a text sequence [57]. There are other pre-trained models, e.g., GloVe [67] and word2vec [59], that can be used for deriving the FEs. We favor embeddings derived from BERT (and SBERT), because these embeddings are contextual and known to better capture sequence-level semantics, including referential relations, when compared to the (non-contextual) embeddings from GloVe and word2vec [48]. In Section 4.5, we experiment with three different ways of deriving FEs from BERT.

For a triple $\langle c_j, p_j, a_{jk} \rangle$, the intermediate output of the ML-based solutions is a predicted label that assumes one of the following three values: *correct* (meaning that p_j refers to a_{jk}), *incorrect* (meaning that p_j does not refer to a_{jk}) or *inconclusive* (meaning that the referential relation between p_j and a_{jk} is not clear enough to be classified as either *correct* or *incorrect*). $\text{ML}_{ensemble}$ generates its intermediate output by combining the predictions from ML_{LF} and ML_{FE}. If ML_{LF} and ML_{FE} agree on the label predicted for a given triple, then $\text{ML}_{ensemble}$ assigns this label to the triple as well. If ML_{LF} and ML_{FE} disagree, then $\text{ML}_{ensemble}$ assigns to the triple the label predicted with the higher probability, but only if the difference between the two probabilities is greater than or equal to a threshold θ_δ. If the probability difference falls short of θ_δ, then $\text{ML}_{ensemble}$ assigns the label *inconclusive*. Based on our tuning presented in Section 4.5, we recommend $\theta_\delta = 0.1$.

For ambiguity detection, we train the classifiers underlying our ML-based solutions on a subset of the *DAMIR* dataset, with datapoints covering all three outcome classes (*correct*, *incorrect* and *inconclusive*). Doing so enables the classifiers to distinguish unambiguous cases (*correct* and *incorrect*) from ambiguous ones (*inconclusive*). For anaphora resolution, we train the classifiers on only the datapoints labeled *correct* or *incorrect*. For this task, the datapoints labeled *inconclusive* are not useful and may even mislead the learning of correct and incorrect referential relations.

The rules used by our ML-based solutions for ambiguity handling are inspired by Yang et al. [95] and provided in Table 1. There is a threshold θ_β in the rules for controlling the detection results. We recommend $\theta_\beta = 0.5$, based on the tuning results of Section 4.5. To illustrate the ML-based solutions, recall the example of Figure 2. For that example, the input would be five triples: $\langle c_1, p_1, a_{1k} \rangle; 1 \leq k \leq 5$. For ambiguity detection, when trained and tuned as we explain in Section 4.5, ML_{LF} predicts *inconclusive* for all triples, whereas ML_{FE} predicts *inconclusive* for $k \in \{1, 2, 5\}$ and *incorrect* for the rest. These predictions jointly lead to $\text{ML}_{ensemble}$ predicting *inconclusive* for all triples. Due to space, we do not show and argue through the probability scores that $\text{ML}_{ensemble}$ uses for deriving its results for our illustrative example. When the ambiguity-handling rules are applied to these intermediate results, none of the ML-based solutions provide a resolution for p_1, and all three detect p_1 as *ambiguous*.

(iii) Solution based on NLP coreference resolvers. We refer to our final solution, numbered ⑥ in Figure 3, as NLP_{coref}. This solution requires two independent coreference resolvers and can easily be implemented using the NLP pipeline. Let us denote the two resolvers by $Coref_1$ and $Coref_2$. NLP_{coref}, as shown in Table 1, combines the results of $Coref_1$ and $Coref_2$ via consensus. We instantiate $Coref_1$ and $Coref_2$ using two popular coreference resolvers [14]: the resolver in the CoreNLP toolkit [15, 16] and the one in the SpaCy library [31]. For the example of Figure 2, NLP_{coref} resolves p_1 as referring to a_{14} ("the request"), thus deeming p_1 as *unambiguous*.

4 EMPIRICAL EVALUATION

In this section, we tune and assess the alternative solutions presented in Section 3.

4.1 Research Questions (RQs)

Our evaluation tackles the following three research questions (RQs):

RQ1. Which solution alternative is the most accurate for detecting anaphoric ambiguity in requirements? By comparing the accuracy of the alternative solutions in Figure 3, we identify, in RQ1, the best-performing solution for detecting anaphoric ambiguity in requirements.

RQ2. Which solution alternative is the most accurate for resolving anaphora in requirements? In RQ2, we identify among the alternatives in Figure 3, the one that is most accurate for resolving anaphora. Having an accurate anaphora resolver is beneficial for RE in at least two ways: First, during requirements reviews, the machine-generated interpretations are a good indicator for the risk of misunderstandings. Notably, if the requirements analyst(s) settle on an interpretation that differs from the one (if any) offered by automated resolution, then there is an increased chance that other stakeholders, e.g. developers, may misinterpret the anaphora in question, with this misinterpretation potentially happening much later in the development process and thus potentially being more costly to fix. Second, for automated information extraction purposes, e.g., the extraction of conceptual models from requirements [8, 79], one would typically want to use the results of automated anaphora resolution as-is and without additional manual processing.

Table 2: Summary Statistics for our Datasets.

		DAMIR	ReqEval	CoNLL2011
Unique Sentences		1,251	98	6,888
Pronouns	Ambiguous	342	62	-
	Unambiguous	395	47	6,757
Triples	Correct	404	66	6,866
	Incorrect	2,814	104	14,666
	Inconclusive	3,448	272	-

RQ3. What is the execution time of each solution alternative? Execution time is an important factor for ensuring practicality. RQ3 examines the execution time of each of the alternatives in Figure 3.

4.2 Implementation and Availability

We use Python 3.8 [89] for implementing the preprocessing step (Section 3.2) as well as for the high-level scripting of the alternative solutions shown in Figure 3. The NLP pipeline and language-feature extraction are implemented using SpaCy 3.0.5 [31], NLTK 3.5 [49], Stanza 1.2 [73], and CoreNLP 4.2.2 [51]. The SpanBERT-based solutions use the Transformers 4.6.1 library [92] provided by Hugging Face (https://huggingface.co/) and operated in PyTorch [65]. For the ML-based solutions, we use Scikit-learn 0.24.1 [66]. We use the Transformers library for extracting embeddings. BERT's embeddings are extracted from the *bert-base-cased* model. For extracting SBERT's embeddings, we use the *paraphrase-MPNet-base-v2* model [85], also available in the Transformers library. Finally, for implementing the solution that uses existing NLP resolvers, we use the coreference resolution modules available in SpaCy 3.0.5 [31] and CoreNLP 4.2.2 [15, 16]. The different solutions proposed in this paper are implemented using Jupyter Notebooks [42].

4.3 Datasets

We use three datasets in our evaluation. The first dataset has been curated using two external (non-author) annotators, as we elaborate momentarily. We call this dataset *DAMIR*, which stands for **D**ataset for **A**naphoric a**M**biguity **I**n **R**equirements. The other two datasets are borrowed from the literature. These are *CoNLL2011* [34, 53, 72], the NLP dataset on coreference resolution released in the 2011 edition of the Computational Natural Language Learning conference (CoNLL2011); and *ReqEval* [1], the RE dataset on anaphoric ambiguity released in the 2020 edition of the NLP4RE workshop. We use the *CoNLL2011* dataset for fine-tuning the SpanBERT-based solutions. We use the *ReqEval* dataset to evaluate the solution alternatives. The *DAMIR* dataset is split into two portions, as we explain later; one portion is used for development and tuning, and the other portion is used for evaluating the solution alternatives.

Table 2 provides summary statistics for *DAMIR* and the adapted versions of *CoNLL2011* and *ReqEval*. Specifically, the table shows the number of unique sentences in each dataset, the number of pronouns marked as *ambiguous* and *unambiguous*, and the number of triples marked as *correct*, *incorrect* and *inconclusive*. We discuss the three datasets next. Note that the number of correct antecedents is greater than the number of unambiguous pronouns since the correct antecedent can occur in the context multiple times, in which case it will be counted more than once.

DAMIR. We collected 22 industrial requirements specifications (RSs) from eight application domains: satellite communications, medicine, aerospace, security, digitization, automotive, railway, and defence. The requirements in these specifications were independently analyzed by two third-party annotators with expertise in linguistics. The first annotator, who has a Masters degree in cultural studies and multilingualism, had, prior to her engagement in our work, done a six-month internship, focusing on investigating the linguistic characteristics of requirements. The second annotator has a computer-science background with a Masters degree in quality management. This annotator further has a professional certificate in English translation. Both annotators received training on anaphoric ambiguity in requirements. The annotators' work spanned two months, with a total of 44 and 56 hours declared by the annotators, respectively. To mitigate fatigue effects, the annotators were encouraged to limit their periods of work to two hours at a time. In addition to the original RSs, we shared with the annotators the lists of automatically generated triples ($\langle c_j, p_j, a_{jk} \rangle$).

The annotators were asked to examine the list of triples associated with each pronoun occurrence p_j. If they were confident that a candidate antecedent a_{jk} is the likely one in a triple, then they were instructed to label that triple as *correct* and all other triples involving p_j as *incorrect*. In case of doubt, the annotators were asked to label all the triples involving p_j as *inconclusive*. The annotators could also select the label *invalid* if some automatically generated triple had an error caused by, e.g., inaccurate splitting of the sentence constituents. All such invalid triples were filtered out.

To construct the DAMIR dataset, we checked the annotations for the triples associated with each p_j. If the annotators agreed that a triple should be labeled as *correct* (meaning that they also agreed that the other triples for p_j should be labeled as *incorrect*), we considered p_j as *unambiguous*. In this case, the triples associated with p_j received the same labels as indicated by the annotators. If the annotators disagreed on the label for any triple associated with p_j, then we regarded p_j as *ambiguous*, and consequently, labeled all the associated triples as *inconclusive*. We identified two types of disagreement between the annotators: (i) one annotator found p_j *ambiguous* and labeled its triples as *inconclusive*, while the other annotator found p_j *unambiguous* and labeled some triple as *correct*; or (ii) the annotators labeled two different triples as *correct*, i.e., they unconsciously disagreed on the interpretation. We define as an agreement any case other than (i) and (ii) above. Using Fleiss' kappa (κ) [27], we obtain an inter-rater agreement of $\kappa = 0.54$, which indicates moderate agreement [45] between the annotators. We note that for datasets related to ambiguity analysis, this level of agreement is to be expected [20], considering that disagreements are indicators for ambiguous cases.

We split the pronoun occurrences in *DAMIR* into two disjoint subsets: $DAMIR^T$ and $DAMIR^E$. The contexts for the elements in $DAMIR^T$ are also distinct from those for the elements in $DAMIR^E$, i.e., all triples associated with a pronoun p_j including the candidate antecedents a_{jk} of p_j appear in either $DAMIR^T$ or $DAMIR^E$ but not in both. $DAMIR^T$ contains 80% of the dataset and is used for developing and tuning the solutions. $DAMIR^E$ contains the remaining

20% and is used for evaluation. Our empirical evaluation, presented in Section 4, is conducted using $DAMIR^E$ only.

CoNLL2011. We extracted from the original *CoNLL2011* dataset only the annotations relevant to anaphoric ambiguity analysis, i.e., the annotations where a pronoun has been labeled with the antecedent it refers to. We used the source documents released alongside *CoNLL2011* in order to identify a context of size two for each pronoun occurrence. To adapt this dataset to our work, we generated $\langle c_j, p_j, a_{jk} \rangle$ triples through preprocessing (Section 3.2). We then assigned labels to the triples in a backward manner: A triple $\langle c_j, p_j, a_{jk} \rangle$ is labeled *correct* if a_{jk} represents the selected antecedent for p_j. Otherwise, the triple is labeled *incorrect*. We note that no triple is marked as *inconclusive* here, since *CoNLL2011* was not created for ambiguity detection; all pronoun occurrences in *CoNLL2011* are regarded as *unambiguous*.

ReqEval. The *ReqEval* dataset is composed of a set of independent requirements, each with at least one pronoun occurrence. Each pronoun occurrence is labeled as either *ambiguous* or *unambiguous*. In the latter case, the correct antecedent is provided. To adapt *ReqEval* to our work, we generated $\langle c_j, p_j, a_{jk} \rangle$ triples through preprocessing. In contrast to the *DAMIR* and *CoNLL2011* datasets where we set the context size to two when generating the triples, for *ReqEval*, we use a context of size one. This is because we could not ascertain that the requirements were in any particular order; a context beyond the immediate sentence where a pronoun appears was not intended in *ReqEval*. For each *ambiguous* p_j, we assigned the label *inconclusive* to all triples associated with p_j. For each *unambiguous* p_j, we assigned the label *correct* to the triple where a_{jk} matches the antecedent provided for p_j and *incorrect* to all other triples.

4.4 Evaluation Metrics

Anaphoric ambiguity detection. We evaluate ambiguity detection using *precision (P)*, *recall (R)* and F_β*-score* computed as $P = TP/(TP+FP)$, $R = TP/(TP+FN)$, and $F_\beta = (1+\beta^2)(P*R)/(\beta^2 * P + R)$, respectively. A *true positive (TP)* is a case where the solution correctly predicts p_j as ambiguous. A *true negative (TN)* is a case where the solution correctly predicts p_j as unambiguous. A *false positive (FP)* is a case where the solution falsely predicts p_j as ambiguous, and a *false negative (FN)* is a case where the solution falsely predicts p_j as unambiguous. As is common for many requirements analysis tasks including ambiguity analysis [11, 95], we favor recall over precision. We thus use and report F_2-scores (i.e., $\beta = 2$).

Anaphora resolution. We evaluate resolution using the following metric, which we call *success rate*: the ratio of correctly resolved pronoun occurrences to the total number of pronoun occurrences labeled as unambiguous in the ground truth. We apply two modes to decide whether the antecedent identified by a solution is correct as per the ground truth. In the *full matching* mode, we consider the identified antecedent to be correct only when it fully matches the text span in the ground truth. In the *partial matching* mode, we consider the identified antecedent to be correct if it overlaps with the text span in the ground truth. For example, the identified antecedent "all approval requests" compared to "approval requests" (in the ground truth) is considered as correctly resolved in partial matching but not in full matching. The rationale for considering partial matching is that, when the matching results are destined for a manual review, pinpointing the location of the text span of interest is highly useful, even though the identified span may be incomplete or only partially correct.

4.5 Solutions Tuning

In this section, we describe the tuning of our solutions. The resulting tuned solutions are used in Section 4.6 for answering RQ1-3.

Tuning SpanBERT. We fine-tune the SpanBERT-based solutions to maximize F_2-score for ambiguity detection. We follow the recommendations in the literature for fine-tuning pre-trained language models [18, 36, 68]. To generate SpanBERT_{NLP} (solution ① in Figure 3), we fine-tune SpanBERT on the *CoNLL2011* dataset for 20 epochs with 2e-5 learning rate and 32 batch size. We then generate SpanBERT_{RE} (solution ② in Figure 3) by fine-tuning SpanBERT_{NLP} for 3 epochs on the $DAMIR^T$ dataset with the same learning rate and batch size as used in solution ①.

We apply a threshold θ_α as the lower bound for accepting a text span identified by solution ① or ② as the antecedent of a pronoun occurrence (see Section 3.3). We tune θ_α on $DAMIR^T$ via exhaustive search. Specifically, we experiment with 10 values $[0.1, 0.2, \cdots, 1.0]$. The optimal value is $\theta_\alpha = 0.9$.

Tuning ML. We optimize ML_{LF} and ML_{FE} (solutions ③ and ④ in Figure 3) on $DAMIR^T$. We consider different configurations that arise from varying the ML classification algorithm and the FEs. For both ③ and ④, we experiment with seven widely used classification algorithms, namely decision tree (DT), feedforward neural network (FNN), k-nearest neighbor (kNN), logistic regression (LR), naïve Bayes (NB), random forest (RF) and support vector machine (SVM) [50, 74, 95]. Following best practice [18, 77], we explore four options for extracting FEs for solution ④. In the first option, FE_1, the embeddings are extracted from SBERT. The other three options, FE_2–FE_4, are based on embeddings from BERT. FE_2 are the embeddings from the second-to-last hidden layer; FE_3 are the concatenation of the embeddings from the last four hidden layers; and FE_4 are the summation of the embeddings from these four layers.

The various options explained above induce seven configurations for solution ③ and 28 for solution ④. We tune solutions ③ and ④ for maximizing F_2-score for ambiguity detection in $DAMIR^T$. We further tune the solutions for maximizing the success rate of anaphora resolution (using only the datapoints labeled *correct* or *incorrect*, and excluding those labeled *inconclusive*). Since *correct* is the minority class in anaphora resolution, we downsize the *incorrect* class using random under-sampling [35].

We evaluate all configurations using 10-fold cross-validation [91]. We note that standard 10-fold cross-validation would partition $DAMIR^T$ at the triple level, implying that some of the triples associated with a pronoun occurrence could land in the training set and the others in the test set. Such splitting of the triples associated with the same pronoun is undesirable. We therefore develop a variant of 10-fold cross-validation where we first group the datapoints in $DAMIR^T$ by pronoun occurrence, perform random shuffling and only then split the dataset into ten equal partitions. This ensures that all the triples associated with a single pronoun occurrence are placed in one partition only, used either for training or for testing.

Tables 3 and 4 list the various configurations and the results obtained for each. We note that, in Table 4, we apply the *full matching*

Table 3: Accuracy of Different Configurations of Solutions ③ and ④ for Anaphoric Ambiguity Detection.

		DT			FNN			kNN			LR			NB			RF			SVM		
		P	R	F_2	P	R	F_2	P	R	F_2	P	R	F_2	P	R	F_2	P	R	F_2	P	R	F_2
③	*LF*	**50.9**	94.0	80.3	50.2	96.2	81.2	50.3	91.0	78.3	49.5	89.0	76.6	50.4	**99.7**	**83.3**	49.9	94.3	80.0	50.0	94.4	80.1
④	FE_1	51.0	86.3	75.7	51.6	99.2	83.6	50.2	98.2	82.4	50.3	95.3	80.6	50.8	98.3	82.8	50.3	94.5	80.3	50.0	97.2	81.7
	FE_2	49.8	89.0	76.7	51.1	98.0	82.7	49.8	96.8	81.4	50.4	95.9	81.2	50.1	96.9	81.5	51.1	96.7	82.0	50.4	97.7	82.2
	FE_3	51.9	89.0	77.7	49.9	94.0	79.8	50.2	97.7	82.1	51.2	97.2	82.3	50.6	98.7	82.9	50.6	95.5	80.9	50.4	97.7	82.2
	FE_4	56.9	87.3	77.5	55.5	96.9	83.6	55.7	98.3	84.7	52.5	90.4	78.3	55.0	95.3	82.8	**57.7**	**100**	**85.9**	56.0	92.6	80.3

† FE_1: FEs from SBERT, FE_2: FEs from BERT's second-to-last layer, FE_3: concatenation of FEs from BERT's last four layers, FE_4: summation of FEs from the same four layers.

Table 4: Success Rate of Different Configurations of Solutions ③ and ④ for Anaphora Resolution.

		DT	FNN	kNN	LR	NB	RF	SVM
③	*LF*	32.2	81.4	61.0	86.4	18.6	71.1	**91.5**
④	FE_1	6.8	**74.6**	13.6	66.1	62.7	66.1	69.4
	FE_2	0.0	59.3	16.9	66.1	55.9	67.8	71.1
	FE_3	8.5	61.0	16.9	66.1	18.6	67.8	66.1
	FE_4	6.8	57.6	15.2	62.7	52.5	57.6	66.1

mode for computing accuracy. This is because the ML-based solutions predict an exact candidate antecedent from a pre-generated list instead of demarcating a text span. The best results for each solution are highlighted in **bold**. We select as the best-performing configuration for ML_{LF} the *NB* algorithm for ambiguity detection, and the *SVM* algorithm for anaphora resolution. We select as the best-performing configuration for ML_{FE} the *RF* algorithm trained over FE_4 for ambiguity detection, and the *FNN* algorithm trained over FE_1 for anaphora resolution. Following the above decisions, we apply grid search [9] to optimize the hyperparameters of the best-performing configurations; hyperparameter optimization for all possible configurations would have been too expensive due to the high dimensionality of feature embeddings.

Finally, there are two fixed thresholds in the ML-based solutions, θ_β and θ_δ, which we tune after hyperparameter optimization. The role of θ_β is the same as that of θ_α, discussed earlier for the SpanBERT-based solutions. The θ_β threshold is tuned in the same manner as θ_α. The optimal value is $\theta_\beta = 0.5$. As for θ_δ, the threshold is used by $ML_{ensemble}$ to ensure that one candidate antecedent is not favored over another when the predicted probabilities are too close (see Section 3.3). We tune θ_δ using exhaustive search on $DAMIR^T$ and over the same ten values tried for θ_α and θ_β. The optimal value is $\theta_\delta = 0.1$.

4.6 Answers to the RQs

RQ1. *Which solution alternative is the most accurate for detecting anaphoric ambiguity in requirements?* Table 5 (left side) shows the precision (P), recall (R) and F_2-score (F_2) of the different solutions measured on the $DAMIR^E$ and *ReqEval* datasets.

As shown by the table, all alternatives perform better on *ReqEval* than $DAMIR^E$. The difference in accuracy is particularly notable for the precision of SpanBERT-based solutions. We believe that this difference can be explained by the different context sizes used for pronoun occurrences in the two datasets. In *ReqEval*, the context is one sentence with an average length of 25 words, where both the pronouns and their antecedents occur. In this dataset, the average number of candidate antecedents for a pronoun is four. In contrast, in $DAMIR^E$, the context is composed of two sentences with an average of 47 words. For this dataset, the average number of candidate antecedents is nine, i.e., more than twice as many as for *ReqEval*. Parsing larger contexts and having to deal with more candidate antecedents allow more room for error. Overall, we believe that the results for $DAMIR^E$ are more reflective of practice, since analysts often consider a broader context for a pronoun than the sentence where the pronoun appears. As noted earlier, this broader context information is unavailable in *ReqEval*, hence our evaluation using single sentences as context in this dataset.

As seen from Table 5, the ML-based solutions have the best recall (and also precision) on both datasets. We believe that the superior accuracy of the ML-based solutions has to do with the fact that these solutions are explicitly trained to distinguish ambiguous and unambiguous pronoun occurrences. We further observe that the choice of features in ML-based solutions, i.e., LFs versus FEs, has little impact on the accuracy of ambiguity detection. Overall $ML_{ensemble}$ leads to the best F_2-scores, including perfect recall on both datasets. In terms of precision, the ML-based solutions are the superior ones as well. We note that ML_{LF} and ML_{FE} neither achieve perfect recall on *ReqEval* nor offer tangible gains over $ML_{ensemble}$ in terms of precision. Across *ReqEval* and $DAMIR^E$, $ML_{ensemble}$ has an average precision of 59.9%. We believe that this level of precision is acceptable in practice. The implication of a ≈60% precision is the manual effort needed for filtering out the pronouns wrongly marked as ambiguous (FPs). Discarding FPs is still easier and requires less effort than finding FNs, i.e., the ambiguous cases that are missed.

> The answer to **RQ1** is that *$ML_{ensemble}$ (solution ⑤ in Figure 3) with an average precision of ≈60% and a recall of 100% is the most accurate solution for detecting anaphoric ambiguity in requirements.*

RQ2. *Which solution alternative is the most accurate for resolving anaphora in requirements?* Table 5 (right side) shows the resolution success rate (defined in Section 4.4) for $DAMIR^E$ and *ReqEval*. Our evaluation covers 96 unambiguous pronouns in $DAMIR^E$ and 62 in

Table 5: Accuracy Results for Different Anaphoric Ambiguity Handling Solutions.

	Precision, Recall and F_2 of Ambiguity Detection (**RQ1**)						Success Rate (%) of Anaphora Resolution (**RQ2**)			
	DAMIR^E			ReqEval			DAMIR^E		ReqEval	
	P (%)	R (%)	F_2 (%)	P (%)	R (%)	F_2 (%)	Full	Partial	Full	Partial
① SpanBERT$_{NLP}$	40.0	81.8	67.6	60.2	75.8	72.1	73.5	88.2	**97.8**	97.8
② SpanBERT$_{RE}$	36.9	77.2	63.3	57.6	96.8	85.2	68.9	**96.1**	**97.8**	**100**
③ ML$_{LF}$	57.6	**100**	87.2	61.8	98.9	88.3	81.2	-	57.4	-
④ ML$_{FE}$	57.5	**100**	87.1	**62.2**	98.2	88.0	51.0	-	82.9	-
⑤ ML$_{ensemble}$	**58.2**	**100**	**87.5**	61.5	**100**	**88.9**	**82.3**	-	57.4	-
⑥ NLP$_{Coref}$	52.4	48.5	49.2	56.7	51.5	52.5	52.5	52.5	39.6	51.7

† For each dataset, the best values of P, R, F_2 and success rate are highlighted in **bold**.

ReqEval. We apply both the full and partial matching modes (see Section 4.4). We note though that only full matching applies to the ML-based solutions, since these solutions identify the antecedent of a pronoun from a pre-calculated list of candidate antecedents.

As Table 5 shows, for anaphora resolution, no solution outperforms all the others on both datasets. For instance, the ML-based solutions (③ – ⑤) perform well on one dataset but not the other. ML$_{ensemble}$ is the best-performing solution on *DAMIR*^{E}, but performs rather poorly on *ReqEval*. As highlighted in the table, the SpanBERT-based solutions clearly outperform all other solutions in partial matching mode, with SpanBERT$_{RE}$ achieving the highest success rate. We thus believe that SpanBERT$_{RE}$ is the most useful solution in terms of providing assistance to analysts during manual requirements reviews.

> The answer to **RQ2** is that *SpanBERT$_{RE}$ (solution ② in Figure 3) with an average success rate of ≈98% is the most accurate solution for resolving anaphora in requirements.*

RQ3. *What is the execution time of each solution alternative?* We consider the execution time of our solutions both from the perspective of a solution developer and that of an end-user.

A developer would be interested in how long it takes to tune the SpanBERT- and ML-based solutions, as discussed in Section 4.5. Tuning is a *one-off activity* and not pertinent to end-users. We used Google Colaboratory [12] for developing and tuning the SpanBERT-based solutions. Fine-tuning SpanBERT on *CoNLL2011* (with 6,757 pronouns) to generate SpanBERT$_{NLP}$ took ≈4 hours. Fine-tuning SpanBERT$_{NLP}$ on *DAMIR*^{T} (with 533 pronouns) to generate SpanBERT$_{RE}$ took ≈23 minutes. For tuning the ML-based solutions, we used a workstation equipped with a 12-core processor (AMD Ryzen 9 5900X 3.7 GHz) and 64 GB of memory. Recall from Section 4.5 that the ML-based solutions are tuned separately for ambiguity detection and anaphora resolution. Tuning time is directly impacted by the best-performing configuration picked for each task (which will then be subjected to hyperparamater optimization). Tuning ML$_{LF}$ required 30 minutes for detection and 53 minutes for resolution. Tuning ML$_{FE}$ was more expensive, requiring 6.5 hours for detection and 45 minutes for resolution.

To measure execution time from an end-user's perspective, we used a normal laptop with a 2.3 GHz CPU and 16 GB of memory. We picked from our evaluation set a random selection of 100 pronoun occurrences. These occurrences span 96 requirements sentences and induce 842 triples. We combined the 96 sentences into a single document. The resulting document is not meant to represent a real-world RS. Rather, we want this document to emulate a representative situation for pronoun occurrences (e.g., in terms of having different pronoun types and different numbers of candidate antecedents in context). In a real setting, before one applies any of our solutions to an RS, all the material in the RS other than the sentences within the context of some pronoun occurrence can be removed.

The resulting document was used for measuring *per-pronoun* execution time. The measured times are representative for larger samples as well, with the overall execution time increasing linearly as the number of pronoun occurrences increases.

> The answer to **RQ3** is as follows. *The average time (in seconds) required for handling an individual pronoun occurrence is: 1.5s using SpanBERT$_{NLP}$ or SpanBERT$_{RE}$; 8s for detection and 8s for resolution using ML$_{LF}$; 7.5s for detection and 6s for resolution using ML$_{FE}$; 14.5s for detection and 13s for resolution using ML$_{ensemble}$; and 7s using NLP$_{Coref}$.*

The practical implication of these execution times is as follows: Based on the literature [25], one can expect that 20% of the requirements in a given RS would contain (pronominal) anaphora. Processing a large RS with, say, 2000 requirements would then require processing 400 (give or take) requirements sentences. Extrapolating from our datasets, one can expect 1.2 pronouns per sentence and thus 480 pronouns in our hypothetical RS with 2000 requirements. Using the most accurate solutions from RQ1 and RQ2, one would require about 2 hours for detecting ambiguity using ML$_{ensemble}$ and about 4 minutes for resolving anaphora using SpanBERT$_{RE}$. The execution time of ambiguity detection can be cut by almost half if one applies either ML$_{LF}$ or ML$_{FE}$, potentially at the cost of a slight decrease in recall. These execution times are acceptable for offline processing, e.g., during a break or overnight. As for online (i.e., interactive) processing, we observe that, at any given time, an analyst likely works on only a small fragment of a large document. For interactive usage, anaphoric ambiguity handling can be localized to the document segment (e.g., sentences) that the analyst is working on.

4.7 Discussion

Below, we make two remarks: the first one is the overall conclusion of our empirical evaluation; the second one is a lesson learned about using pre-trained language models in RE.

(1) Given the accuracy results (RQ1 and RQ2) and the execution times (RQ3), we propose a *hybrid* solution for handling anaphoric ambiguity in requirements. This hybrid solution combines supervised ML for ambiguity detection and SpanBERT for anaphora resolution. For the detection task, $ML_{ensemble}$ is the most accurate. One may nonetheless elect to use the slightly less accurate ML_{LF} or ML_{FE} to reduce execution time. For the resolution task, we recommend $SpanBERT_{RE}$. This solution is highly accurate in pinpointing the location of antecedents.

(2) We benefited from the *CoNLL2011* dataset for the initial fine-tuning of SpanBERT, before further fine-tuning it with RE-specific data. Our preliminary experimentation indicated that, without the intermediate fine-tuning step over *CoNLL2011*, SpanBERT would not lead to a viable solution through fine-tuning on our RE datasets alone. We believe that, due to the general scarcity of tailor-made datasets for RE tasks, one should take into account the possibility that intermediate fine-tuning data may be required, when attempting to design requirements automation solutions based on pre-trained language models. To this end, RE researchers may need to look for complementary datasets in other communities, e.g., NLP, to be able to get the best traction from pre-trained language models.

5 THREATS TO VALIDITY

The validity concerns most pertinent to our evaluation are internal and external validity.

Internal Validity. The main concern regarding internal validity is bias. This concern applies mainly to the *DAMIR* dataset, which was developed on the authors' initiative. To mitigate bias, the labelling of *DAMIR* was performed exclusively by two independent (non-author) annotators. To avoid learning bias, the annotators were never exposed to either the design or the results of any of the alternative solutions in our study.

External Validity. We evaluated all solutions on two datasets – *DAMIR* and *ReqEval*, the latter being an external dataset. The individual solutions show comparable results across the two datasets. In terms of domain coverage, *DAMIR* spans eight different application domains. The consistency of the results across the *DAMIR* and *ReqEval* datasets, taken alongside the domain coverage of *DAMIR*, provides confidence about the generalizability of our empirical findings. That said, further evaluation using additional documents and user studies can help further mitigate external-validity threats.

6 CONCLUSION

In this paper, we developed and evaluated six alternative automation solutions for handling anaphoric ambiguity in requirements. Each solution addresses both the detection of anaphoric ambiguity as well as the resolution of anaphora. Our motivation for conducting a multi-solution study stems from the availability of competing NLP and ML technologies that we could build on. Without an empirical examination of different solution designs, we would not be able to ascertain which technologies would be the most suitable for our analytical needs. This situation is not limited to our work per se; choosing the right set of technologies for the task at hand is a consideration that one increasingly has to contend with in AI-enabled automation.

Our evaluation involved two datasets with a total of ≈1,350 industrial requirements. Our results indicate that, for anaphoric ambiguity detection, supervised ML is more accurate than both SpanBERT (a variant of BERT) and a solution built using off-the-shelf coreference resolvers. Our best solution for ambiguity detection has an average precision of ≈60% and a recall of 100%. Differently from the ambiguity detection task, for anaphora resolution, SpanBERT yields the best solution with an average success rate of ≈98%. Based on these results, we recommend a hybrid solution for anaphoric ambiguity handling, where ambiguity detection and anaphora resolution are realized using different technological platforms.

Anaphoric ambiguity is an important but still a single aspect of the broader problem of ambiguity. In requirements engineering, where ambiguity handling is closely associated with quality assurance, analysts are likely interested in a more holistic treatment that addresses a wider range of ambiguity types. In the future, we would like to expand our work to other ambiguity types, particularly semantic ones, that are still under-explored. Furthermore, and to more conclusively evaluate the usefulness of our current results, we plan to conduct user studies involving practicing engineers.

Acknowledgement. This work was funded by Luxembourg's National Research Fund (FNR) under the grant BRIDGES18/IS/12632261 and NSERC of Canada under the Discovery and Discovery Accelerator programs. We are grateful to the research and development team at QRA Corp. for valuable insights and assistance.

REFERENCES

[1] Sallam Abualhaija, Davide Fucci, Fabiano Dalpiaz, and Xavier Franch. 2020. Preface: 3rd Workshop on Natural Language Processing for Requirements Engineering (NLP4RE'20). In *Joint Proceedings of REFSQ-2020 Workshops, Doctoral Symposium, Live Studies Track, and Poster Track co-located with the 26th International Conference on Requirements Engineering: Foundation for Software Quality.*

[2] Sallam Abualhaija, Davide Fucci, Fabiano Dalpiaz, Xavier Franch, and Alessio Ferrari. 2020. ReqEval: The shared task on anaphora ambiguity detection and disambiguation. https://github.com/frieden84/nlp4re-reqeval last accessed: July 2021.

[3] Charu C Aggarwal. 2018. *Machine learning for text.* Springer.

[4] Vincenzo Ambriola and Vincenzo Gervasi. 2006. On the Systematic Analysis of Natural Language Requirements with CIRCE. *Automated Software Engineering* 13, 1 (2006).

[5] Chetan Arora, Mehrdad Sabetzadeh, Lionel Briand, and Frank Zimmer. 2015. Automated Checking of Conformance to Requirements Templates Using Natural Language Processing. *IEEE Transactions on Software Engineering (TSE'15)* 41, 10 (2015).

[6] Chetan Arora, Mehrdad Sabetzadeh, Lionel Briand, and Frank Zimmer. 2017. Automated Extraction and Clustering of Requirements Glossary Terms. *IEEE Transactions on Software Engineering* 43, 10 (2017).

[7] Chetan Arora, Mehrdad Sabetzadeh, Lionel Briand, Frank Zimmer, and Raul Gnaga. 2013. RUBRIC: A Flexible Tool for Automated Checking of Conformance to Requirement Boilerplates. In *Proceedings of the 9th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE'13).*

[8] Chetan Arora, Mehrdad Sabetzadeh, Shiva Nejati, and Lionel Briand. 2019. An Active Learning Approach for Improving the Accuracy of Automated Domain Model Extraction. *ACM Transactions on Software Engineering and Methodology* 28, 1 (2019).

[9] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for hyper-parameter optimization. *Advances in neural information processing systems* 24 (2011).

[10] D. Berry, E. Kamsties, and M. Krieger. 2003. From Contract Drafting to Software Specification: Linguistic Sources of Ambiguity, A Handbook. http://se.uwaterloo.ca/~dberry/handbook/ambiguityHandbook.pdf

[11] Daniel M Berry. 2021. Empirical evaluation of tools for hairy requirements engineering tasks. *Empirical Software Engineering* 26, 6 (2021).

[12] Ekaba Bisong. 2019. *Building machine learning and deep learning models on Google cloud platform: A comprehensive guide for beginners.* Apress.

[13] Samuel Broscheit, Massimo Poesio, Simone Paolo Ponzetto, Kepa Joseba Rodriguez, Lorenza Romano, Olga Uryupina, Yannick Versley, and Roberto Zanoli. 2010. BART: A multilingual anaphora resolution system. In *Proceedings of the 5th international workshop on semantic evaluation.*

[14] Xinyun Cheng, Xianglong Kong, Li Liao, and Bixin Li. 2020. A Combined Method for Usage of NLP Libraries Towards Analyzing Software Documents. In *International Conference on Advanced Information Systems Engineering.*

[15] Kevin Clark and Christopher D. Manning. 2016. Deep Reinforcement Learning for Mention-Ranking Coreference Models. In *Empirical Methods on Natural Language Processing.*

[16] Kevin Clark and Christopher D. Manning. 2016. Improving Coreference Resolution by Learning Entity-Level Distributed Representations. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics.*

[17] Fabiano Dalpiaz, Ivor van der Schalk, Sjaak Brinkkemper, Fatma Aydemir, and Garm Lucassen. 2019. Detecting terminological ambiguity in user stories: Tool and experimentation. *Information and Software Technology* 110 (2019).

[18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of deep bidirectional transformers for language understanding. (2018). arXiv:arXiv:1810.04805

[19] Richard Evans. 2001. Applying machine learning toward an automatic classification of it. *Literary and linguistic computing* 16, 1 (2001), 45–58.

[20] Saad Ezzini, Sallam Abualhaija, Chetan Arora, Mehrdad Sabetzadeh, and Lionel C Briand. 2021. Using domain-specific corpora for improved handling of ambiguity in requirements. In *2021 IEEE/ACM 43rd International Conference on Software Engineering.*

[21] Fabrizio Fabbrini, Mario Fusani, Stefania Gnesi, and Giuseppe Lami. 2001. The linguistic approach to the natural language requirements quality: Benefit of the use of an automatic tool. In *Proceedings of the 26th Annual NASA Goddard Software Engineering Workshop.*

[22] Henning Femmer, Daniel Méndez Fernández, Elmar Juergens, Michael Klose, Ilona Zimmer, and Jörg Zimmer. 2014. Rapid requirements checks with requirements smells: Two case studies. In *Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering.*

[23] Henning Femmer, Daniel Méndez Fernández, Stefan Wagner, and Sebastian Eder. 2017. Rapid quality assurance with Requirements Smells. *Journal of Systems and Software* 123 (2017).

[24] Alessio Ferrari and Andrea Esuli. 2019. An NLP approach for cross-domain ambiguity detection in requirements engineering. *Automated Software Engineering* 26, 3 (2019).

[25] Alessio Ferrari, Gloria Gori, Benedetta Rosadini, Iacopo Trotta, Stefano Bacherini, Alessandro Fantechi, and Stefania Gnesi. 2018. Detecting requirements defects with NLP patterns: An industrial experience in the railway domain. *Empirical Software Engineering* 23, 6 (2018).

[26] Alessio Ferrari, Giorgio Oronzo Spagnolo, and Stefania Gnesi. 2017. Pure: A dataset of public requirements documents. In *2017 IEEE 25th International Requirements Engineering Conference.*

[27] Joseph L. Fleiss. 1971. Measuring nominal scale agreement among many raters. *Psychol. Bull.* 76, 5 (1971).

[28] Vincenzo Gervasi, Alessio Ferrari, Didar Zowghi, and Paola Spoletini. 2019. Ambiguity in Requirements Engineering: Towards a Unifying Framework. In *From Software Engineering to Formal Methods and Tools, and Back.* Springer.

[29] Benedikt Gleich, Oliver Creighton, and Leonid Kof. 2010. Ambiguity Detection: Towards a Tool Explaining Ambiguity Sources. In *Proceedings of the 16th Working Conference on Requirements Engineering: Foundation for Software Quality.*

[30] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning* (1st ed.). MIT Press.

[31] Matthew Honnibal, Ines Montani, Sofie Van Landeghem, and Adriane Boyd. 2020. *spaCy: Industrial-strength Natural Language Processing in Python.* https://doi.org/10.5281/zenodo.1212303

[32] Mitra Bokaei Hosseini, Rocky Slavin, Travis Breaux, Xiaoyin Wang, and Jianwei Niu. 2020. Disambiguating Requirements Through Syntax-Driven Semantic Analysis of Information Types. In *Proceedings of the 26th Working Conference on Requirements Engineering: Foundation for Software Quality.*

[33] Yufang Hou. 2020. Bridging Anaphora Resolution as Question Answering. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics.*

[34] David Graff Huang, Shudong and George Doddington. 2002. *Multiple-Translation Chinese Corpus LDC2002T01.* Web download file. Philadelphia: Linguistic Data Consortium.

[35] Nathalie Japkowicz. 2000. The class imbalance problem: Significance and strategies. In *Proceedings of the International Conference on Artificial Intelligence.*

[36] Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S Weld, Luke Zettlemoyer, and Omer Levy. 2020. SpanBERT: Improving pre-training by representing and predicting spans. *Transactions of the Association for Computational Linguistics* 8 (2020).

[37] Dan Jurafsky and James H. Martin. 2020. *Speech and Language Processing* (3rd ed.). https://web.stanford.edu/~jurafsky/slp3/(visited 2021-06-04).

[38] Erik Kamsties. 2005. *Understanding Ambiguity in Requirements Engineering.* Springer Berlin Heidelberg.

[39] Erik Kamsties and Barbara Peach. 2000. Taming ambiguity in natural language requirements. In *Proceedings of the 13th International Conference on Software and Systems Engineering and Applications.*

[40] Nadzeya Kiyavitskaya, Nicola Zeni, Luisa Mich, and Daniel Berry. 2008. Requirements for tools for ambiguity identification and measurement in natural language requirements specifications. *Requirements Engineering* 13, 3 (2008).

[41] Pohl Klaus and Rupp Chris. 2011. *Requirements Engineering Fundamentals* (1st ed.). Rocky Nook.

[42] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Damián Avila, Safia Abdalla, and Carol Willing. 2016. Jupyter Notebooks – a publishing format for reproducible computational workflows. In *Positioning and Power in Academic Publishing: Players, Agents and Agendas.*

[43] Giuseppe Lami, Mario Fusani, and Gianluca Trentanni. 2019. QuARS: A Pioneer Tool for NL Requirement Analysis. In *From Software Engineering to Formal Methods and Tools, and Back.* Springer.

[44] Matthew Lamm, Jennimaria Palomaki, Chris Alberti, Daniel Andor, Eunsol Choi, Livio Baldini Soares, and Michael Collins. 2021. Qed: A framework and dataset for explanations in question answering. *Transactions of the Association for Computational Linguistics* 9 (2021).

[45] J. Richard Landis and Gary G. Koch. 1977. An Application of Hierarchical Kappa-type Statistics in the Assessment of Majority Agreement among Multiple Observers. *Biometrics* 33, 2 (1977).

[46] Kusum Lata, Pardeep Singh, and Kamlesh Dutta. 2021. A comprehensive review on feature set used for anaphora resolution. *Artificial Intelligence Review* 54, 4 (2021).

[47] Timothy Lee, Alex Lutz, and Jinho D Choi. 2016. QA-It: classifying non-referential it for question answer pairs. In *Proceedings of the ACL 2016 Student Research Workshop.*

[48] Qi Liu, Matt J Kusner, and Phil Blunsom. 2020. A survey on contextual embeddings. (2020). arXiv:arXiv:2003.07278

[49] Edward Loper and Steven Bird. 2002. NLTK: The Natural Language Toolkit. In *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics.*

[50] Panos Louridas and Christof Ebert. 2016. Machine Learning. *IEEE Software* 33, 5 (2016).

[51] Christopher Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven Bethard, and David McClosky. 2014. The Stanford CoreNLP Natural Language Processing Toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations.*

[52] Mitch Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics* 19, 2 (1993).

[53] Murray Grossman Nii Martey John Bell Mark Liberman, Kelly Davis. 2002. *Emotional Prosody Speech and Transcripts LDC2002S28.* CD-ROM. Philadelphia: Linguistic Data Consortium.

[54] Aaron Massey, Richard Rutledge, Annie Anton, and Peter Swire. 2014. Identifying and classifying ambiguity for regulatory requirements. In *Proceedings of the 22nd IEEE International Requirements Engineering Conference.*

[55] Alistair Mavin, Philip Wilkinson, Adrian Harwood, and Mark Novak. 2009. Easy Approach to Requirements Syntax (EARS). In *Proceedings of the 17th IEEE International Requirements Engineering Conference.*

[56] Joseph F McCarthy and Wendy G Lehnert. 1995. Using Decision Trees for Coreference Resolution. In *International Joint Conferences on Artificial Intelligence.*

[57] Alessio Miaschi and Felice Dell'Orletta. 2020. Contextual and Non-Contextual Word Embeddings: an in-depth Linguistic Investigation. In *Proceedings of the 5th Workshop on Representation Learning for NLP.* Association for Computational Linguistics.

[58] L. Mich. 1996. NL-OOPS: From natural language to object oriented requirements using the natural language processing system LOLITA. *Natural Language Engineering* 2, 2 (1996).

[59] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. (2013). arXiv:arXiv:1301.3781

[60] Ruslan Mitkov. 1999. *Anaphora resolution: the state of the art.* Citeseer.

[61] Ruslan Mitkov. 2014. *Anaphora resolution.* Routledge.

[62] Natalia N Modjeska, Katja Markert, and Malvina Nissim. 2003. Using the web in machine learning for other-anaphora resolution. In *Proceedings of the 2003 conference on Empirical methods in natural language processing.*

[63] Ray Offen. 2002. Domain understanding is the key to successful system development. *Requirements engineering* 7, 3 (2002).

[64] Mohamed Osama, Aya Zaki-Ismail, Mohamed Abdelrazek, John Grundy, and Amani Ibrahim. 2020. Score-based automatic detection and resolution of syntactic

ambiguity in natural language requirements. In *2020 IEEE International Conference on Software Maintenance and Evolution*.

[65] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc.

[66] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[67] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation. In *Empirical Methods in Natural Language Processing*.

[68] Jason Phang, Thibault Févry, and Samuel R Bowman. 2018. Sentence encoders on stilts: Supplementary training on intermediate labeled-data tasks. (2018). arXiv:arXiv:1811.01088

[69] Steven Piantadosi, Harry Tily, and Edward Gibson. 2012. The communicative function of ambiguity in language. *Cognition* 122, 3 (2012).

[70] Massimo Poesio, Roland Stuckardt, and Yannick Versley. 2016. *Anaphora resolution*. Springer.

[71] Klaus Pohl. 2010. *Requirements Engineering* (1st ed.). Springer.

[72] Sameer Pradhan, Lance Ramshaw, Mitch Marcus, Martha Palmer, Ralph Weischedel, and Nianwen Xue. 2011. CoNLL-2011 shared task: Modeling unrestricted coreference in ontonotes. In *Proceedings of the Fifteenth Conference on Computational Natural Language Learning: Shared Task*. 1–27.

[73] Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton, and Christopher D. Manning. 2020. Stanza: A Python Natural Language Processing Toolkit for Many Human Languages. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*.

[74] Felipe Quecole, Maisa Cristina Duarte, and Estevam Rafael Hruschka. 2018. Coupling for Coreference Resolution in a Never-ending Learning System. *Journal of Information and Data Management* 9, 2 (2018).

[75] Karthik Raghunathan, Heeyoung Lee, Sudarshan Rangarajan, Nathanael Chambers, Mihai Surdeanu, Dan Jurafsky, and Christopher D Manning. 2010. A multi-pass sieve for coreference resolution. In *Proceedings of the 2010 conference on empirical methods in natural language processing*.

[76] Marta Recasens, Lluís Màrquez, Emili Sapena, M Antònia Martí, Mariona Taulé, Véronique Hoste, Massimo Poesio, and Yannick Versley. 2010. Semeval-2010 task 1: Coreference resolution in multiple languages. In *Proceedings of the 5th International Workshop on Semantic Evaluation*.

[77] Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. (2019). arXiv:arXiv:1908.10084

[78] Cristina Ribeiro and Daniel Berry. 2020. The prevalence and severity of persistent ambiguity in software requirements specifications: Is a special effort needed to find them? *Science of Computer Programming* 195 (2020).

[79] Marcel Robeer, Garm Lucassen, Jan Martijn E.M. van der Werf, Fabiano Dalpiaz, and Sjaak Brinkkemper. 2016. Automated Extraction of Conceptual Models from User Stories via NLP. In *Proceedings of the 24th IEEE International Requirements Engineering Conference*.

[80] Danissa Rodriguez, Doris Carver, and Anas Mahmoud. 2018. An efficient wikipedia-based approach for better understanding of natural language text related to user requirements. In *Proceedings of the 39th IEEE Aerospace Conference*.

[81] Benedetta Rosadini, Alessio Ferrari, Gloria Gori, Alessandro Fantechi, Stefania Gnesi, Iacopo Trotta, and Stefano Bacherini. 2017. Using NLP to Detect Requirements Defects: An Industrial Experience in the Railway Domain. In *Proceedings of the 23rd Working Conference on Requirements Engineering: Foundation for Software Quality*.

[82] Chetan Arora Mehrdad Sabetzadeh Saad Ezzini, Sallam Abualhaija. 2021. *"Online Annex (online)"*. Available at https://tinyurl.com/2p9k2zf2, August 2021.

[83] Nicolas Sannier, Morayo Adedjouma, Mehrdad Sabetzadeh, and Lionel Briand. 2017. An automated framework for detection and resolution of cross references in legal texts. *Requirements Engineering* 22, 2 (2017).

[84] Unnati Shah and Devesh Jinwala. 2015. Resolving Ambiguities in Natural Language Software Requirements: A Comprehensive Survey. *SIGSOFT Software Engineering Notes* 40, 5 (2015).

[85] Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. 2020. Mpnet: Masked and permuted pre-training for language understanding. (2020). arXiv:arXiv:2004.09297

[86] Rhea Sukthanker, Soujanya Poria, Erik Cambria, and Ramkumar Thirunavukarasu. 2020. Anaphora and coreference resolution: A review. *Information Fusion* 59 (2020).

[87] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. 2016. *Introduction to data mining*. Pearson Education India.

[88] Sri Tjong and Daniel Berry. 2013. The design of SREE—a prototype potential ambiguity finder for requirements specifications and lessons learned. In *Proceedings of the 19th Working Conference on Requirements Engineering: Foundation for Software Quality*.

[89] Guido Van Rossum and Fred L. Drake. 2009. *Python 3 Reference Manual*. CreateSpace.

[90] Yawen Wang, Lin Shi, Mingyang Li, Qing Wang, and Yun Yang. 2020. A Deep Context-wise Method for Coreference Detection in Natural Language Requirements. In *2020 IEEE 28th International Requirements Engineering Conference*.

[91] Ian Witten, Eibe Frank, Mark Hall, and Christopher Pal. 2011. *Data Mining: Practical Machine Learning Tools and Techniques* (4th ed.). Elsevier.

[92] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-Art Natural Language Processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Association for Computational Linguistics.

[93] Wei Wu, Fei Wang, Arianna Yuan, Fei Wu, and Jiwei Li. 2020. CorefQA: Coreference resolution as query-based span prediction. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*.

[94] Hui Yang, Anne De Roeck, Vincenzo Gervasi, Alistair Willis, and Bashar Nuseibeh. 2010. Extending nocuous ambiguity analysis for anaphora in natural language requirements. In *Proceedings of the 18th IEEE International Requirements Engineering Conference*. IEEE.

[95] Hui Yang, Anne de Roeck, Vincenzo Gervasi, Alistair Willis, and Bashar Nuseibeh. 2011. Analysing anaphoric ambiguity in natural language requirements. *Requirements Engineering* 16, 3 (2011).