UNIVERSITÉ DU LUXEMBOURG

PhD-FSTM-2022-096
The Faculty of Science, Technology and Medicine

DISSERTATION

Defence held on 05/09/2022 in Luxembourg

to obtain the degree of

DOCTEUR DE L'UNIVERSITÉ DU LUXEMBOURG

EN INFORMATIQUE

by

Saad EZZINI
Born on 19 December 1994 in Bensouda FEZ (Morocco)

ARTIFICIAL INTELLIGENCE-ENABLED AUTOMATION FOR AMBIGUITY HANDLING AND QUESTION ANSWERING IN NATURAL-LANGUAGE REQUIREMENTS

Dissertation defence committee

Dr Sallam Abualhaija, Dissertation Supervisor
Research Scientist, Université du Luxembourg

Dr Yves Le Traon, Chairman
Professor, Université du Luxembourg

Dr Bissyande Tegawende, Vice-Chairman
Associate Professor, Université du Luxembourg

Dr Andreas Vogelsang, Member
Professor, University of Cologne, Germany

Dr Davide Fucci, Member
Assistant Professor, Blekinge Institute of Technology, Sweden

Acknowledgement

I would like to express my gratitude to everyone who helped and supported me on this four years journey. I am deeply indebted to my supervisors Mehrdad Sabetzadeh and Sallam Abualhaija, who introduced me to the software engineering field and provided countless help from the beginning to the very end of this journey.
I would like to thank the committee members for agreeing to review my thesis and attend my defense. Especially external members who had to travel to Luxembourg.
I am also grateful to my co-authors and colleagues in the SVV lab and SnT. I would be remiss not to mention the industrial partnership we had with QRA and the valuable insights and assistance provided by their research and development team during the first half of my PhD. Additionally, special thanks to Luxembourg's National Research Fund, European Research Council, and Natural Sciences and Engineering Research Council of Canada, who financed my research, for their generous support.
This endeavor would not have been possible without my family, including my wife and children, for encouraging me and providing support and motivation, and my parents for their encouragement throughout my career.

Saad EZZINI
University of Luxembourg
September 2022

Abstract

Requirements engineering (RE) quality control is a crucial step for a project's success. Natural language (NL) is by far the most commonly used means for capturing requirement specifications. Despite facilitating communication, NL is prone to quality defects, one of the most notable of which is ambiguity. Ambiguous requirements can lead to misunderstandings and eventually result in a system that is different from what is intended, thus wasting time, money, and effort in the process. This dissertation tackles selected quality issues in NL requirements:

- **Using Domain-specific Corpora for Improved Handling of Ambiguity in Requirements:** Syntactic ambiguity types occurring in coordination and prepositional-phrase attachment structures are prevalent in requirements (in our document collection, as we discuss in Chapter 3, 21% and 26% of the requirements are subject to coordination and prepositional-phrase attachment ambiguity analysis, respectively). We devise an automated solution based on heuristics and patterns for improved handling of coordination and prepositional-phrase attachment ambiguity in requirements. As a prerequisite for this research, we further develop a more broadly applicable corpus generator that creates a domain-specific knowledge resource by crawling Wikipedia.

- **Automated Handling of Anaphoric Ambiguity in Requirements: A Multi-solution Study:** Anaphoric ambiguity is another prevalent ambiguity type in requirements. Estimates from the RE literature suggest that nearly 20% of industrial requirements contain anaphora [1, 2]. We conducted a multi-solution study for anaphoric ambiguity handling. Our study investigates six alternative solutions based on three different technologies: (i) off-the-shelf natural language processing (NLP), (ii) recent NLP methods utilizing language models, and (iii) machine learning (ML).

- **AI-based Question Answering Assistant for Analyzing NL Requirements:** Understanding NL requirements requires domain knowledge which is not necessarily shared by all the involved stakeholders. We develop an automated question-answering assistant that supports requirements engineers during requirements inspections and quality assurance. Our solution uses advanced information retrieval techniques and machine reading comprehension models to answer questions from the same requirement specifications document and/or an external domain-specific knowledge resource.

All the research components in this dissertation are tool-supported. Our tools are released with open-source licenses to encourage replication and reuse.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Context

Requirements engineering (RE) is a sub-field of software engineering (SE) that addresses the elicitation and specification of the goals, desired characteristics, capabilities, and constraints of software-intensive systems [3, 4]. The quality of requirements has a direct impact on the overall success of a project. In particular, ensuring the precision and consistency of requirements is paramount for avoiding major development risks such as time and budget overruns, failure to meet users' needs, and systems that are not trustworthy. Examples where poor requirements have had serious negative consequences are plenty. To give one, in 2000, FBI started to develop a case management system, named the Virtual Case File. The project failed due to inadequate requirements with a cost implication of over $200 million [5].

A software requirements specification (SRS) is considered as a central artifact in RE, laying out the requirements of the system-to-be [6]. An SRS is typically intended for a diverse audience, e.g., product managers, domain experts, and developers. To facilitate communication among stakeholders, the overwhelming majority of SRSs are written in natural language (NL) [7]. A key advantage of NL is that it facilitates shared understanding among different stakeholders who may have different backgrounds and expertise, often requiring little or no additional training [8]. Despite this advantage, NL requirements are prone to a variety of quality issues, including different types of ambiguity [9, 10, 11], incompleteness [12], and inconsistency. Requirements are often manually inspected to identify such quality issues.

This dissertation aims to provide AI-based automated support for helping requirements engineers with inspecting NL requirements more efficiently and identifying different quality issues, with a focus on ambiguity. Ambiguity is an inherent phenomenon in NL, occurring when a text segment is open to multiple interpretations [13]. Manual ambiguity handling in requirements is challenging. Not only is this task time-consuming and error-prone, but it also requires, to a certain level, domain knowledge that facilitates interpreting requirements, consequently highlighting the ambiguous ones. SRSs vary across domains and thus use a domain-specific vocabulary [9]. Incorporating domain-knowledge into automated ambiguity handling solutions is therefore advantageous in order to obtain more accurate results.

In the RE literature, ambiguity in NL requirements (among other quality issues) has been extensively

studied [14, 15, 16, 17, 18]. In an early work, the different ambiguity types pertinent to NL requirements are discussed [19]. Following this, several methods have been proposed in the literature concerning ambiguity detection in requirements [20, 21, 9, 22]. Existing work in RE has five limitations.

C1. Despite ambiguity being prevalent in requirements, there is little work in RE dedicated to handling the syntactic ambiguity types addressed in this dissertation, namely coordination, prepositional-phrase attachment, and anaphoric ambiguity.

C2. Current research in RE focuses exclusively on detecting ambiguity in requirements without providing recommendations about their interpretations.

C3. Existing domain-specific methods for detecting ambiguity in requirements cannot be easily extended to domains other than the ones used in development. No automated means are provided to adapt the same solution for addressing ambiguity across diverse domains.

C4. Automated solutions using natural language processing (NLP) do not exploit the full potential of NLP. Recent advances in the NLP field, notably the dominant use of large-scale language models such as BERT [23], prompt both new RE automation solutions as well as a reexamination of the existing ones.

C5. Related to **C4**, the fifth limitation is that question answering (QA) has been studied in RE with a limited scope [24, 25, 26, 27], and to the best of our knowledge, has not yet been explored in the context of requirements quality assurance.

Our work in this dissertation takes steps toward addressing the above limitations. To address **C1**, we provide automated support for improved handling of the syntactic ambiguity types mentioned above. Complementing the existing evidence in the RE literature about the prevalence of ambiguity [28, 19, 29, 30, 31, 21, 1, 32], we observe that 21%, 26%, and 25% out of 5000 industrial requirements investigated in our work are subject to coordination, prepositional-phrase attachment, and anaphoric ambiguity analysis, respectively. To address **C2** and **C3**, we incorporate domain-specific knowledge that is automatically adapted to the underlying domain of the SRS under analysis in providing recommendations about the interpretations of requirements. This in turn leads to more accurate ambiguity detection. To address **C4** we devise multiple new solutions and re-implement existing ones to assess their effectiveness in the context of NL requirements. Our solutions cover a spectrum of technologies including off-the-shelf NLP tools, feature-based machine learning (ML) methods and the recent language models. Finally, with regard to **C5**, we devise a QA assistant that facilitates inspecting SRSs, where requirements engineers can pose questions in NL concerning quality issues (e.g., incomplete requirements) and domain-specific interpretations of requirements (e.g., clarification of certain equations or definitions of domain concepts).

1.2 Contributions and Organization

As shown in Figure 1.1, in this dissertation, we present five components structured in three chapters as follows:

- **Chapter 3: Using Domain-specific Corpora for Improved Handling of Ambiguity in Requirements.** In this chapter, we propose an automated approach for improving the handling of coordination ambiguity (CA) and prepositional-phrase attachment ambiguity (PAA) in NL requirements. CA can potentially occur when the two conjuncts are preceded or followed by a modifier [29]. PAA can occur when a

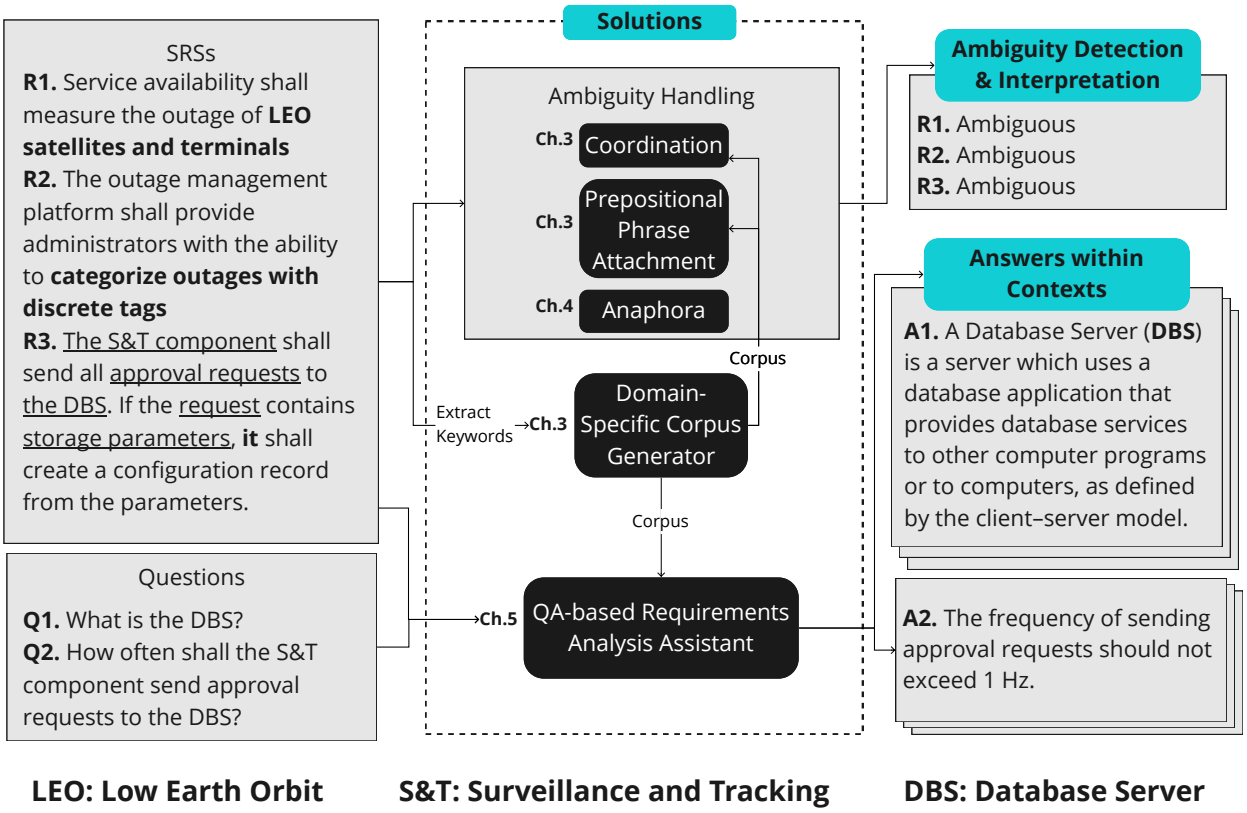


Figure 1.1: Dissertation Overview.

prepositional-phrase (PP) is preceded by a verb and a noun phrase, then it becomes unclear whether the PP is an adverbial modifier (attached to the preceding verb) or a noun attribute (attached to the preceding noun phrase) [28]. The requirement R1 in Figure 1.1 contains CA with two possible interpretations. The first interpretation occurs when the modifier "LEO" (low-earth orbit) modifies the two conjuncts "satellites" and "terminals". The second interpretation occurs when the modifier "LEO" modifies "satellites" only. The requirement R2 in Figure 1.1 contains PAA with two possible interpretations due to the presence of a prepositional-phrase (PP) attachment. The first interpretation occurs when the PP "with discrete tags" is attached to the verb. The second interpretation occurs when the PP is attached to the noun "outages". Our approach builds on and enhances an ensemble of structural patterns alongside heuristics from the existing literature. An important novelty aspect in our CA and PAA ambiguity handling approach is the use of domain-specific corpora, and since these corpora cannot be assumed to be available a priori in most cases, we take steps to build such corpora automatically. To this end, we develop a novel approach that automatically extracts domain-specific corpora from Wikipedia and utilizes them to increase the accuracy of CA and PAA handling in requirements documents. We conduct a large-scale evaluation of our approach using more than 5000 industrial requirements from seven different application domains. Based on our evaluation, our approach can detect CA and PAA with an average precision of ≈80% and an average recall of ≈89% (≈90% for cases of unacknowledged ambiguity). This work has been published in a conference paper [10] and an artifact paper [33].

Concretely, our contributions in Chapter 3 are as follows:

– We propose and devise an automated solution that leverages NLP and ML for improved handling of

CA and PAA.

- We implement our approach in Java and release it for public use [33] alongside the datasets used for building the approach.
- We implement the domain-specific corpus generator (a sub-component of our overall improved ambiguity handling approach) and release it for public use. Specifically, we re-implement this sub-component in Python to be more compatible with the recent NLP literature. Further details are provided in the tool support section 3.5.2 and submitted as a tool paper [34].
- Large-scale evaluation of our approach on a set of industrial requirements from diverse application domains.

- **Chapter 4: Automated Handling of Anaphoric Ambiguity in Requirements: A Multi-solution Study.** In this chapter, we develop, evaluate, and compare six alternative automated solutions for anaphoric ambiguity handling in NL requirements. We focus in our work on pronominal anaphoric ambiguity (most relevant anaphoric type to RE) which occurs when there is more than one plausible antecedent to which an anaphor (pronoun) can refer [35, 9]. Estimates from the RE literature suggest that nearly 20% of industrial requirements contain anaphora [1, 2]. The requirement R3 in Figure 1.1 contains pronominal anaphoric ambiguity. Here, the anaphor is *it*, the potential antecedents are the preceding noun phrases (NPs), namely "the S&T component", "approval requests", "the DBS", "the request" and "storage parameters". It is not clear which subsystem should create a configuration record in this case. Motivated by identifying the most accurate technology platform, we systematically compare multiple alternatives to achieve our goal. We devise our solutions based on language models, machine learning (ML), different learning features, and NLP off-the-shelf tools. Each solution addresses both the detection of anaphoric ambiguity and the resolution of anaphora. Our evaluation involved two datasets with a total of ≈1,350 industrial requirements. The best-performing solution for anaphoric ambiguity detection is the one based on ML which achieved an average precision of ≈60% and a recall of 100%. The best-performing solution for anaphora resolution is the one based on language models, it achieved a success rate of ≈98% in interpreting anaphora. This work has been published in a conference paper [11] and a follow-up tool paper has been submitted [36], and is included in the tool support section 4.5.

  Concretely, our contributions in Chapter 4 are the following:

  - We conduct a comparative analysis that empirically evaluates six alternative solutions across different available technologies for improved handling of anaphoric ambiguity.
  - We publicly release the Dataset for Anaphoric aMbiguity In Requirements (DAMIR), which we created and annotated for this task.
  - We implement all considered solutions, and the hybrid best-performing solution emerged in our study and release it for public use.

- **Chapter 5: AI-based Question Answering Assistant for Analyzing Natural-language Requirements.** In this chapter, we devise *ReQAssis*, an AI-based automated quality assurance assistance by means of QA. *ReQAssis* takes as input an SRS and a question posed in NL. *ReQAssis* then employs two models: a *Retriever* and a *Reader* that jointly provide the requirements engineer with the output consisting of a list of relevant text passages and a potential answer highlighted in each passage for a given question.

Figure 1.1 shows the use case example of posing a question about the definitions of a domain-specific term, "the DBS" in this case. The approach automatically generates external domain-specific knowledge resource by crawling Wikipedia and finds the likely answer to the question. Our approach also enables questions related to quality issues within a given SRS. For example, the question "How often shall the S&T component send approval requests to the DBS?" is aimed at verifying whether the requirement R3 is complemented by further details in another requirement in the same SRS. To evaluate our approach, we generated the Requirements Engineering Question-Answering dataset (*REQuestA*) that is specific to NL requirements in a semi-automatic manner. REQuestA is created from a collection of six SRSs covering three different application domains, namely aerospace, defence and security. *REQuestA* contains a total of 387 question-answer pairs of which 173 are automatically generated, and the rest are proposed by third-party analysts. We empirically evaluate *ReQAssis* on *REQuestA* dataset. *ReQAssis* can retrieve the relevant document for domain-specific questions from a corpus with 100% accuracy. It can retrieve the text passage that contains the right answer to the input question among the top-3 relevant text passages with an average accuracy of 90.6%, and extract from the right passage the likely answer to the question with an average accuracy of 84%.

Concretely, our contributions in this chapter are as follows:

- We devise an automated QA assistance that supports requirements engineers in inspecting requirements more efficiently.
- We empirically evaluate our approach on a semi-automatically generated dataset.
- We publicly release the RE Question-Answering dataset (REQuestA), which we created and annotated for this task.
- We implement the best configuration of REQAssis, developed in Python, and release it to the public.

Chapter 2

Background

This chapter provides background information for this dissertation. The content of the chapter is organized under three main topics: (1) Natural Language Processing (NLP), (2) Machine Learning (ML), (3) Language Modeling (LM), and (4) Word Vectorization.

2.1 Natural Language Processing

Natural Language Processing (NLP), a sub-field of artificial intelligence (AI), can broadly be defined as the application of computational technologies to process and analyze natural language (NL), that is, written and spoken human language [37].

In our work, we generally employ an NLP pipeline composed of the following components: (1) *Tokenizer* to split the text into tokens, for example, the sentence "what time is it?" is tokenized as five tokens [What, time, is, it,?]; (2) *Sentence Splitter* to break up the text into individual sentences; (3) *Part-Of-Speech (POS) Tagger* to assign a POS tag, e.g., noun, verb, or pronoun, to each token in each sentence; (4) *Lemmatizer* to identify the canonical form (lemma) of each token, for example, the lemma for "playing" is "play"; (5) *Constituency Parser* to identify the structural units of sentences, e.g., verb phrases and noun phrases; (6) *Dependency Parser* for identifying the grammatical dependencies between tokens in sentences, e.g., subject and object; (7) *Coreference Resolver* to find mentions that refer to the same textual entity; and finally, (8) *Semantic Parser* to extract information about the meanings of words.

2.2 Machine Learning

Machine learning (ML), another sub-field of AI, is concerned with developing models that learn on a sample data, called *training data*, to make predictions and decisions on new unseen data (*test data*) [38].

Machine learning systems can be categorized into four types according to the type of supervision they receive during training [39]. (1) *Supervised learning* algorithms require feeding the desired solutions in the training data (*labels*). Supervised learning involves two main tasks, classification and regression. In short, classification deals with categorical targets, whereas regression deals with continuous numerical targets. (2) *Unsupervised*

learning algorithms, on the other hand, do not require training labels and try to learn without a teacher. (3) *Semi-supervised* learning algorithms deal with partially labeled data and mostly unlabeled data, they often combine supervised and unsupervised techniques. (4) *Reinforcement learning* systems are based on agents that observe the environments and perform actions and get a positive or negative reward for each action [39].

In this thesis, we work primarily with supervised techniques. In this context, we briefly discuss the supervised learning models as well some additional concepts that we use and experiment with such as data imbalance handling and ensembling techniques.

2.2.1 Data Imbalance Handling

Data imbalance handling is the process of improving the quality of unbalanced data, where some classes are under-represented and make a small minority of the data compared to other classes. There are two popular ways to balance the data before feeding it to ML models, oversampling and undersampling. Oversampling techniques create new data points of the minority class, and undersampling ones remove some of the majority class data points, until reaching a balance. In the following, we present two sampling techniques, Synthetic Minority Oversampling Technique (SMOTE), and random resampling.

- Synthetic Minority Oversampling Technique (SMOTE) is an oversampling technique that selects two examples from the minority class that are close in the feature space using the k-nearest neighbour technique, then creates a new point in a random location between the selected examples or data points [40].
- Random Resampling groups two simple resampling techniques, Random Oversampling and Random Undersampling [41].
  - Random Oversampling randomly duplicates data points from minority class.
  - Random Undersampling randomly removes data points from the majority class.

2.2.2 Supervised ML algorithms:

In this dissertation, we use the following supervised ML classification algorithms: Decision tree (DT), feed-forward neural network (FNN), k-nearest neighbor (kNN), logistic regression (LR), naïve Bayes (NB), random forest (RF), support vector machine (SVM), AdaBoost (ADA), and XGBoost (XGB).

- Decision Tree (DT) is a supervised learning method that predicts the target value by learning simple rules inferred from data points and characteristics and building a tree structure of decision rules. The DTs are simple to interpret and understand, and they can be visualized. However, they can easily overfit on the training data [42].
- Feed-forward neural network (FNN) is a simple neural network that consists of a number of processing units that are interconnected and organized in layers. The size of the input layer is the number of data features, and the size of the output layer is the number of classes. The FNN is called feed-forward algorithm because it has no feedback between layers, this means that the output of any layer does not affect that same layer or the preceding ones [43].
- k-Nearest Neighbour (kNN) algorithm is a simple supervised learning technique that classifies new data points based on similarity (neighboring) to the available classes of the training set [44].

- Logistic Regression (LR) is a probabilistic supervised classification algorithm that estimates the probability that an instance belong to a certain class. LR provides probabilistic values between 0 and 1 [45].

- Naïve Bayes (NB) is a simple probabilistic algorithm that is based on the Bayes theorem to solve supervised classification problems. The Bayes theorem provides the conditional probability of a first event A, given that a second event B has occurred. NB is an effective algorithm for high-dimensional datasets; it does not require a long training time and provides quick predictions [46].

- Random Forest (RF) is a well-known supervised learning algorithm based on combining multiple decision trees trained on different subsets of the training dataset and predicting the final output based on the majority votes of the predictions [47].

- Support Vector Machine (SVM) is another well-known supervised learning algorithm. The objective of SVM is to create the best decision boundary (hyperplane) that can segregate an n-dimensional space into classes. This hyperplane is created based on extreme points (vectors) which are referred to as support vectors [48].

- AdaBoost (ADA), short for adaptive boosting, is an iterative ensemble boosting classification model (classifier) that builds a strong classifier (provides highly accurate predictions) by combining multiple weak classifiers (performs only slightly better than random guessing) to increase the accuracy of the predictions. In each iteration, ADA sets the weights of the classifiers, ensuring accurate predictions of unusual observations [49].

- XGBoost (XGB), short for Extreme Gradient Boosting, is an optimized implementation of Gradient Boosted Decision Trees (GBDT). GBDT is an iterative ensemble boosting classifier that involves DTs. In each DT iteration, GBDT adjusts the weight, coefficient, or bias values applied to each input variable to predict the target class. In this dissertation, we refer to Gradient Boosted Decision Trees as XGBoost (XGB) [50].

2.2.3 Ensembling techniques

Ensembling in ML can be defined as combining the decisions of individual ML models or reinforcing a main model with smaller models, with the goal of providing an improved performance. There are multiple ensembling techniques; however, we present two main categories that we use over the course of this dissertation: voting and stacking.

- **Voting.** There are two types of voting techniques used to combine ML predictions. Hard voting, known as majority voting, counts the number of votes for each class provided by all participating algorithms. For each data point, the majority wins, which means that the output class is the one voted on by the majority voters. Soft voting uses the prediction probabilities of the participating algorithms. The predictions probabilities of each class are summed, and the predicted class is the one with the greatest sum (gets most of the vote). For both voting types, weights can be assigned to strong algorithms and a preferred class can be set in case of a tie.

- **Stacking** is an advanced voting technique that trains a new model (meta-learner) on the output of multiple models to provide improved overall accuracy. The base models used to obtain the initial predictions are

generally called weak learners. The meta-learner is not exposed to the training dataset, but it takes as input the weak-learner predictions, and it attempts to learn the combination of these predictions to make a better output prediction.

2.3 Language Modeling

Language models are probabilistic deep learning models that are trained to determine the probability of occurrence of a word or a sequence of words based on the surrounding context and the training text/corpora. BERT [23] (Bidirectional Encoder Representations from Transformers) is a notable example of such pre-trained models, which have been widely applied for solving many downstream NLP tasks, such as Named Entity Recognition, Sentiment Analysis, and Text Classification.

BERT is pre-trained on the BooksCorpus [51] and English Wikipedia [52], with two training objectives, namely masked language modeling (MLM) and next sentence prediction (NSP). In MLM, a fraction of the tokens in the pre-training text are randomly masked. The model then learns 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". NSP requires the model to predict a binary label that represents whether two segments are consecutive in the original text. BERT uses the *Transformer* architecture proposed in [53]. A Transformer is a sequence model that has an encoder-decoder structure. The encoder takes an input sequence (say in English) and transforms it into embeddings, whereas the decoder takes the embeddings and creates the output sequence (say, the translated sequence in French). The Transformer draws out the significant power of parallelization. The BERT model architecture applies a 12-layer bidirectional Transformer encoder [23]. The encoder consists of multiple layers, where each layer has two sublayers, namely a multi-head (self-)attention mechanism and a feed-forward neural network. Self-attention is a mechanism that is used to compute the representation of a single sequence. For example, in the text sequence "bank of the river", the words "bank" and "river" influence each other. Therefore, their embeddings can be re-weighed to capture more context concerning the meaning of "bank" that is related to water and not money. NL text is more complicated than the example above and often allows for more than one attention (the meaning of "bank"). To illustrate, consider the example sequence "I gave food to my dog Charlie". We see in this example multiple possible attentions like "Charlie" is "my dog", "I" am the one who "gave", and "food" is given to "Charlie". The multi-head attention mechanism allows the model to jointly attend to information from different representation subspaces.

SpanBERT is a variant of BERT that is optimized for the prediction of spans of text. Unlike BERT, SpanBERT masks random continuous spans, rather than random tokens. SpanBERT is trained only on one objective, that is, the span boundary objective (the start and end of the text span boundary). SpanBERT improves BERT's performance on text span prediction and selection tasks.

2.4 Word Vectorization

Word Vectorization involves transforming single words into numeric representations – also called vectors. Below, we present Term Frequency - Invert Document Frequency (TFIDF), Word2Vec, BERT embeddings, and SentenceBERT embeddings.

Term Frequency - Invert Document Frequency (TFIDF)

Term Frequency - Invert Document Frequency (TF-IDF) is statistical score that reflects the importance of a word to a document in a set of documents. TF-IDF builds a sparse vector representation for each word. The TF-IDF score is the product of the values of TF (term frequency) and IDF (inverse document frequency). TF is computed per query as the frequency count of the query term in the document divided by the total number of terms in the document. IDF is computed per query as the logarithm of the division of the total number of documents by the number of documents containing the query [54].

Word2Vec

Word2Vec (word to vector) builds and groups vectors of similar words allowing to capture the meaning of each word based on the training dataset/corpus. Word2Vec can be built using two main architectures, skip-gram which try to predict surrounding words given a target word, and continuous bag of words (CBOW) which try to predict a single words from a fixed window of words (context) [55].

Contextualized Embedding

Contextualized embeddings are representations that produce token embeddings by considering the context in which the word occurs. In other words, the same token can have different embeddings when it occurs in different contexts [56].

- **BERT Embeddings.** BERT-base model has 12 layers of transformer encoders. Each layer's output for each token can be used as word embeddings to represent that token. There are multiple options to extract token embeddings from BERT such as considering the last layer or combining the embeddings produced in several layers. Some of these options are reported the NLP literature to yield the best results [23, 57]. In our work, we experiment with these options, namely, the output of the last hidden layer, the summation of the output of all 12 layers, the output of the second-to-last hidden layer, the summation of the output of the last four layers, and concatenation of the output of the last four layers.
- **SentenceBERT Embeddings.** Combining word embeddings, as discussed above, to create sentence representations does not lead to accurate results on sentence-related NLP tasks [57]. In our work, we use SentenceBERT (SBERT), which is an alternative method that is trained to provide meaningful sentence embeddings. SBERT is a modification of pre-trained BERT that uses Siamese network and triplet loss [58] techniques to build models with the ability to provide embeddings for the input sentences.

Chapter 3

Using Domain-specific Corpora for Improved Handling of Ambiguity in Requirements

3.1 Motivation and Contributions

Ambiguity has been widely studied in the requirements engineering (RE) literature [32, 59, 19, 60, 12]. Both manual approaches based on reviews and inspections [59, 61], and automated approaches based on natural language processing (NLP) [20, 21, 9, 22], have been proposed for detecting ambiguity in requirements. Some recent works use domain-specific corpora for detecting terms that are likely to be ambiguous due to different meanings across domains [9, 62, 63, 64]. Current research on ambiguity in RE, as we elaborate later, has three main limitations. First, the research focuses exclusively on detecting ambiguity and does not address automated interpretation for requirements in which no genuine ambiguity exists. The lack of automated interpretation impedes further automated analysis, e.g., automated information extraction from requirements [65, 66]. Second, existing methods for detecting domain-specific ambiguity are restricted to identifying merely words with different meanings across domains, and further require the domain of interest to be specified a priori. Finally, while the negative consequences of unacknowledged ambiguity are known in the RE literature [20], the question of how accurately unacknowledged ambiguity can be detected through automated means has never been investigated empirically.

Motivated by addressing the above limitations, we propose an automated approach for improved ambiguity handling – both ambiguity detection and interpretation – in NL requirements. Ambiguity detection is concerned with finding the requirements that are genuinely ambiguous. Interpretation, in contrast, is concerned with providing the most likely meaning where the potential for ambiguity exists, but where there is no ambiguity. Our approach incorporates domain knowledge by automatically generating domain-specific corpora, without any a-priori assumption about the domain. These corpora alongside a set of structural patterns and heuristics are used for handling ambiguity in requirements. In our evaluation, we analyze the impact of domain knowledge on ambiguity handling. We further assess how well our automated approach can detect unacknowledged ambiguity in different domains.

Our work in this chapter concentrates on *coordination ambiguity* and *prepositional-phrase attachment ambiguity* [28, 19, 29], hereafter referred to as CA and PAA, respectively. Targeting these (syntactic) ambiguity types is motivated by their prevalence in NL requirements [30]. In our document collection, as we will discuss later in the chapter, out of 5156 requirements, 1098 (21%) are subject to CA analysis and 1328 (26%) to PAA analysis. Within these, human annotators acknowledged ambiguity or had different interpretations (unacknowledged ambiguity) in ≈57% of the requirements.

Coordination is a structure that links together two sentence elements (called conjuncts) using a coordinating conjunction (e.g., "and" or "or") [67]. CA can potentially occur when the two conjuncts are preceded or followed by a modifier [29]. The sentence could then be interpretable in two ways, depending on whether only the conjunct next to the modifier is being modified or both conjuncts are being modified [19]. Fig. 3.1 shows an example requirement, R1, with two potential interpretations. The first interpretation, *first read*, hereafter, *FR*, occurs when the modifier "LEO" (low-earth orbit) modifies the two conjuncts "satellites" and "terminals" (Fig. 3.1 (a)). The second interpretation, *second read*, hereafter, *SR*, occurs when the modifier "LEO" modifies "satellites" only (Fig. 3.1 (b)).

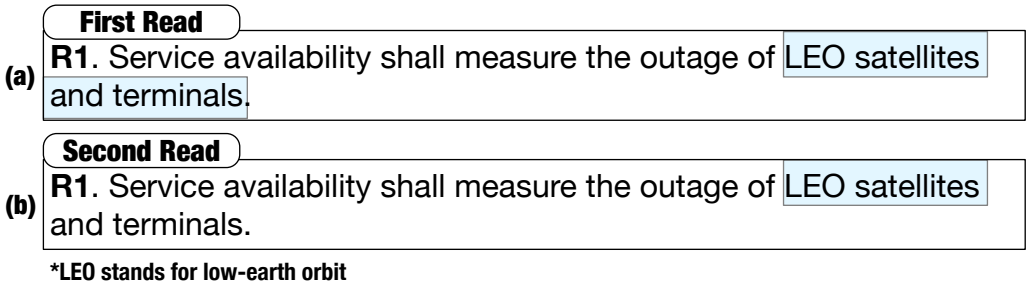


Figure 3.1: Example of Coordination Ambiguity (CA).

A prepositional-phrase (PP) attachment is a PP preceded by a verb and a noun phrase [28]. Virtually all PP attachments have the potential for PAA, because they could be interpretable in two ways, depending on whether the PP is an adverbial modifier (attached to the preceding verb) or a noun attribute (attached to the preceding noun phrase). Fig. 3.2 shows an example requirement, R2, with two potential interpretations due to the presence of a PP attachment. The first interpretation, *verb attachment*, hereafter, *VA*, occurs when the PP "with discrete tags" is attached to the verb "categorize" (Fig. 3.2 (a)). The second interpretation, *noun attachment*, hereafter, *NA*, occurs when the PP is attached to the noun "outages" (Fig. 3.2 (b)).

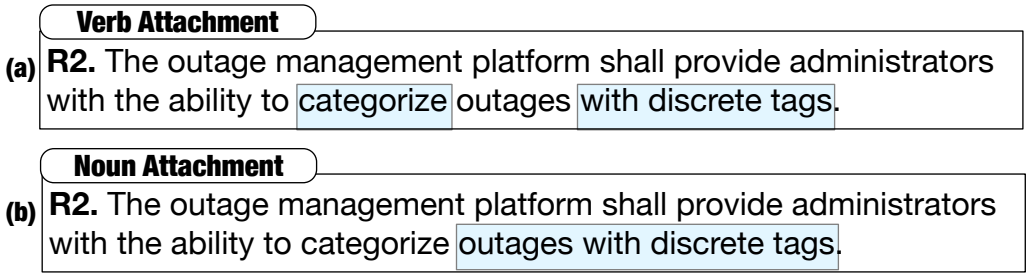


Figure 3.2: Example of Prepositional-phrase Attachment Ambiguity (PAA).

R1 and R2 have the potential to suffer from CA and PAA, respectively. The question is whether these are genuine ambiguities or merely situations that human experts can decisively interpret with little room for divergent interpretations. Existing techniques do not incorporate domain knowledge for providing a likely interpretation of CA; instead, they rely on frequency-based computations derived from a generic corpus [68]. For example,

using existing techniques, attempting to interpret the coordination in R1 would yield *FR*. This interpretation is incorrect; with domain knowledge, the coordination would be interpreted as *SR*. As for the PP attachment in R2, existing techniques are unable to provide an interpretation, although the attachment is interpretable as *VA* with domain knowledge.

Contributions. We take steps toward addressing the limitations outlined above. Our contributions are as follows.

(1) We propose an automated approach for handling CA and PAA in NL requirements. Our approach uses an ensemble of structural patterns and heuristics. Specifically, we match requirements against a set of structural patterns, leveraging and enhancing existing patterns in the literature. In tandem, we attempt to interpret all requirements with coordination and PP-attachment structures using heuristics that are based on semantic, morphological, and frequency information. Some of these heuristics are novel; others are borrowed from the literature and enhanced where necessary. By combining these patterns and heuristics, we attempt to tell apart the requirements that can be disambiguated via automated interpretation from the requirements that are genuinely ambiguous.

(2) We devise a novel domain-specific corpus generator. Without assuming any a-priori knowledge about the domain, we first automatically extract keywords from an input requirements document. Our corpus generator then assembles a large corpus of Wikipedia articles relevant to the terminology (and thus the domain) of the given requirements document. This automatically generated corpus is utilized for increasing the accuracy of the heuristics that rely on frequency-based information. For example, the occurrence of the word "capital" in a requirements document within the aerospace domain differs in frequency and co-occurring words from the same word occurring in a requirements document within the financial domain. Generating and using a domain-specific corpus for ambiguity handling lies at the heart of our proposed approach.

(3) We empirically evaluate our approach on 20 industrial requirements documents. These documents collectively contain 5156 requirements covering seven distinct application domains. The ground truth for our evaluation was prepared by two trained annotators (linguistics experts and non-authors). Our results indicate that: (i) our approach detects CA and PAA with a precision of $\approx$80% and recall of $\approx$89% ($\approx$90% for cases of unacknowledged ambiguity); (ii) the automatic interpretations by our approach have an average accuracy of $\approx$85%; and (iii) using domain-specific corpora leads to substantial gains in accuracy for ambiguity handling, improving detection by an average of $\approx$33% and interpretation by an average of $\approx$16%. We have developed a tool, named MAANA, which implements our approach for the domain-specific handling of ambiguity. Specifically, MAANA detects requirements that potentially contain CA or PAA. The tool and the non-proprietary requirements we use in our evaluation are publicly available at https://github.com/SNTSVV/MAANA.

Structure. Section 3.2 discusses and compares with related work. Section 3.3 presents our approach. Section 3.4 describes our empirical evaluation. Section 3.6 addresses validity considerations. Section 3.7 concludes the chapter.

3.2 Related Work

We focus on handling CA and PAA in NL requirements. Our approach, discussed in Section 3.3, builds on and further enhances the existing structural patterns and heuristics from the RE and NLP literature for CA [69, 70, 71, 72, 73, 74, 75, 68, 76, 77, 78] and PAA [19, 79, 80]. Our work, to our knowledge, is the first to bring these patterns and heuristics together for handling CA and PAA. Below, we position our work against the related work on ambiguity handling in both the RE and NLP communities.

3.2.1 Ambiguity Handling in the RE Community

Ambiguity in requirements has been extensively studied from different perspectives, including understanding the role of ambiguity in RE [81, 82, 32, 83], analyzing the linguistic causes of ambiguity [84, 31, 19, 85], and ambiguity prevention [14, 15, 16, 17, 18]. Automated ambiguity detection solutions in RE are mainly based on matching NL requirements against pre-defined structural patterns using regular expressions, NLP, or both [9]. Numerous approaches and tools have been proposed to this end [60, 86, 21, 75, 17, 87, 1, 2, 88, 89]. In addition to these, some recent works attempt to detect lexical ambiguity – the situation where a word has different meanings depending on the domain [31] – by integrating domain knowledge from Wikipedia [9, 62, 63, 64].

CA detection has been investigated to some extent in the RE literature. Chantree et al. [68] address CA detection using structural patterns and frequency-based heuristics. Their work has been extended over the years [72, 90, 74] with additional heuristics [70, 76], and for anaphora ambiguity detection [21], i.e., ambiguity due to multiple interpretations of pronouns. Though considered a prevalent ambiguity type in requirements [31, 19, 32], to our knowledge, automated handling of PAA has not been previously studied in RE.

Our work differs from or enhances the above research in several ways. First, none of the existing approaches address the automated interpretation of (potentially ambiguous) coordination structures. As for PAA, the topic has not been tackled in RE before. Our approach handles both CA and PAA by combining a broad range of structural patterns and heuristics. Second, none of the existing approaches evaluate the detection of unacknowledged ambiguity. We address this gap in our empirical evaluation. Third, the existing automated methods for domain-specific corpus generation from Wikipedia are limited to a pre-defined set of domains. Our approach, in contrast, can generate a corpus based on any given requirements document without knowing the underlying domain in advance. Fourth and finally, industrial evaluations of ambiguity handling in RE are scarce. Our evaluation contributes to addressing this gap by using a large industrial dataset.

3.2.2 Ambiguity Handling in the NLP Community

Syntactic ambiguity types, including CA and PAA, have been studied for a long time by the NLP community [91]. In an early work by Goldberg [69], CA is handled using conditional probabilities of word co-occurrences. Pantel and Lin [92] present an unsupervised corpus-based method for handling PAA through a notion of contextual similarity. Agirre et al. [79] improve the accuracy of PAA handling by integrating semantic similarity with syntactic parsing. Calvo and Gelbukh [80] propose querying the web for word co-occurrence frequencies and use these frequencies for more accurate PPA handling. In a similar vein, Nakov and Hearst [71] use structural patterns alongside statistical co-occurrence frequencies gathered from the web for handling CA and PAA.

In the context of ambiguity handling, the use of domain knowledge in NLP is mostly directed at word sense disambiguation (WSD) in specific domains [93]. To this end, Wikipedia is a commonly used source of domain knowledge [94, 95]. Fragolli [96] derives from Wikipedia domain-specific corpora as resources for WSD. Similarly, Gella et al. [97] map manually defined topics in WordNet [98, 99] to Wikipedia for generating domain-specific corpora that can in turn be employed for WSD.

We are not aware of any work in the NLP community that uses domain-specific corpora for handling either CA or PAA. Instead, in the existing NLP technologies, e.g., syntax parsing [100], the handling of syntactic ambiguity – CA and PAA included – is tuned over generic texts such as news articles. These technologies therefore do not provide accurate results for CA and PAA in a domain-specific context. As we show in Section 3.4, our approach, which incorporates domain knowledge for handling CA and PAA, provides significant improvements over NLP technologies tuned over generic texts.

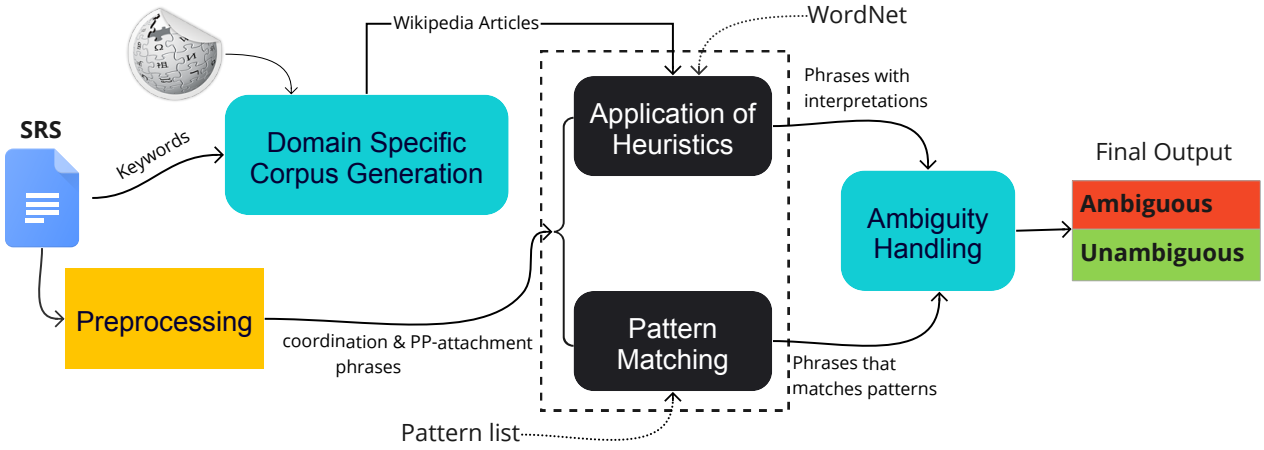


Figure 3.3: Approach Overview.

3.3 Approach

Fig. 3.3 provides an overview of our approach, which is composed of five steps. The input to the approach is an NL requirements document, hereafter, *SRS*. In step 1, we process *SRS* using an NLP pipeline. In this step, we further identify two subsets of the requirements in *SRS*, namely S_c and S_p. These two subsets contain all the requirements with coordination structures and all the requirements with PP attachments, respectively. In step 2, we match the requirements in S_c and S_p against structural patterns that indicate potential CA and PAA, respectively. In step 3, we generate a domain-specific corpus for *SRS* by crawling Wikipedia. Step 3 can be bypassed if a representative corpus for *SRS*'s domain already exists (through earlier applications of our approach to other requirements documents in the same domain). In step 4, we apply a set of heuristics to determine likely interpretations for the requirements in S_c and S_p. In step 5, we classify into ambiguous and unambiguous the requirements in S_c and S_p by combining the results of step 2 and step 4. We note that steps 2 and 4 are independent (i.e., the output of neither step is an input to the other). Step 2 is limited to a finite list of CA and PAA structural patterns. As we will explain later in this section, the heuristics in step 4, when compared to the patterns in step 2, cover a wider spectrum of structures that have the potential for CA and PAA. Combining results from both steps leads to better handling of ambiguity. Below, we elaborate each step of our approach. In the rest of this chapter, ambiguity refers to CA and PAA exclusively.

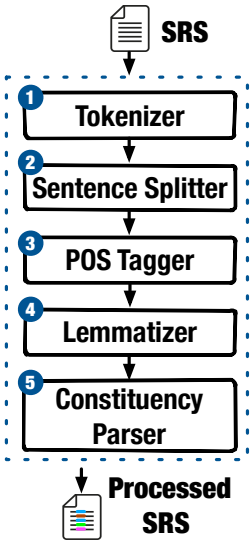
3.3.1 Preprocessing

The NLP pipeline we use for preprocessing *SRS* is depicted in Fig. 3.4. This pipeline is a sequence of five NLP modules. The first module in the sequence, the *Tokenizer*, divides the input text into tokens, such as words and punctuation marks.

The *Sentence Splitter* splits the text into sentences. The *POS Tagger* assigns to tokens part-of-speech (POS) tags, such as noun, verb, and adjective. Next is the *Lemmatizer*, which identifies the canonical form (lemma) for each token. For example, the lemma for "bought" is "buy". Finally, the *Constituency Parser* identifies the structural units of sentences, e.g., noun phrases, verb phrases and prepositional phrases. More details about each component can be found in Chapter 2. The results of the NLP pipeline are used in the next steps.

In this step (step 1), we further identify the two requirements subsets, S_c and

Figure 3.4: NLP Pipeline.



CA	1	nn **n_1** *c* **n_2**	11	**n_1** *c* **n_2** p dt/adj nn	21	adv **adj_1** *c* **adj_2**
	2	**n_1** *c* **n_2** nn	12	**v_1** *c* **v_2** nn	22	**adj_1** *c* **adj_2** adv
	3	nn p **n_1** *c* **n_2**	13	nn p **v_1** *c* **v_2**	23	**adj_1** *c* **adj_2** adj nn
	4	**n_1** *c* **n_2** p nn	14	**v_1** *c* **v_2** p nn	24	nn *dt* **n_1** *c dt* **n_2**
	5	v **n_1** *c* **n_2**	15	v to **v_1** *c* **v_2**	25	nn *p dt* **n_1** *c dt* **n_2**
	6	**n_1** *c* **n_2** v	16	**v_1** *c* **v_2** to v	26	*dt* **n_1** *c dt* **n_2** *p* nn
	7	nn **n_1** *c* **n_2** nn	17	adv **v_1** *c* **v_2**	27	nn *dt* **n_1** *c dt* **n_2** nn
	8	adj **n_1** *c* **n_2**	18	**v_1** *c* **v_2** adv	28	adj *dt* **n_1** *c dt* **n_2**
	9	adj nn **n_1** *c* **n_2**	19	**v_1** *c* **v_2** p dt/adj nn	29	*adj* nn *dt* **n_1** *c dt* **n_2**
	10	adj adj **n_1** *c* **n_2**	20	dt/adj nn p **v_1** *c* **v_2**		

PAA	1	**v n_1** *p* n_2	6	**v n_1** *p dt adj* n_2
	2	**v n_1** *p dt/adj* n_2	7	**v** *n* **n_1** *p dt/adj* n_2
	3	**v** *dt/adj* **n_1** *p* n_2	8	**v** *n* **n_1** *p dt adj* n_2
	4	**v** *dt/adj* **n_1** *p dt/adj* n_2	9	**v** *dt adj* **n_1** *p* n_2
	5	**v** *n* **n_1** *p* n_2	10	**v** *dt adj* **n_1** *p dt/adj* n_2

n_1, n_2, nn: noun, **v**: verb, **adv**: adverb, **adj**: adjective, **dt**: determiner, **p**: preposition, **/**: or.

For CA: **The two conjuncts are in bold** and the modifier is underlined.
For PAA: **The verb and first noun are in bold**, and the second noun is underlined.

Table 3.1: Patterns for ambiguity detection (CA and PAA).

S_p, that should be subject to CA handling and PAA handling, respectively. S_c is comprised of all the requirements in *SRS* that contain "or", "and", or both. We note that only these two conjunctions can lead to CA [68, 29]. S_p is comprised of all the requirements in *SRS* that contain a PP attachment [28]. Requirements that contain a conjunction of interest (i.e., "and" or "or") as well as a PP attachment are included in both S_c and S_p.

3.3.2 Pattern Matching

In this step, we analyze S_c and S_p to identify requirements that are likely to be ambiguous due to their syntactic structure. Table 3.1 lists our patterns for CA and PAA. Of these, 23 CA patterns and four PAA patterns come from the literature [69, 70, 71, 72, 74, 75, 19, 79]. The remaining patterns, shaded blue in the table (i.e., CA patterns #24–29 and PAA patterns #5–10) are novel. The novel patterns were derived by analyzing a subset of the requirements in our dataset, as we will precisely define in Section 3.4.3. Specifically, we analyzed the ambiguous requirements in the *tuning* portion of our dataset.

We match the patterns against the requirements in S_c and S_p. For pattern matching, the unit of analysis is a text *segment*, which is the part of a requirement that matches a given structural pattern from Table 3.1. A pattern suggesting CA matches a segment that contains a conjunction (denoted as c) linking two conjuncts (marked in bold) with a modifier (underlined). For example, the matching segment in R1 (Fig. 3.1) corresponds to pattern#1 for CA, where LEO is the modifier and the conjunction *and* joins the two conjuncts **satellites** and **terminals**. We recall from Section 3.1 that CA occurs when it is unclear whether a modifier is attached to both conjuncts (*FR*) or only to the closest conjunct (*SR*). A pattern suggesting PAA matches a segment with a verb (**v**) followed by a first noun ($\mathbf{n_1}$) – both marked in bold – followed by a PP which consists of a preposition (denoted as p) and a second noun (n_2 – underlined). For example, the matching segment in R2 (Fig. 3.2), "**categorize outages** *with*

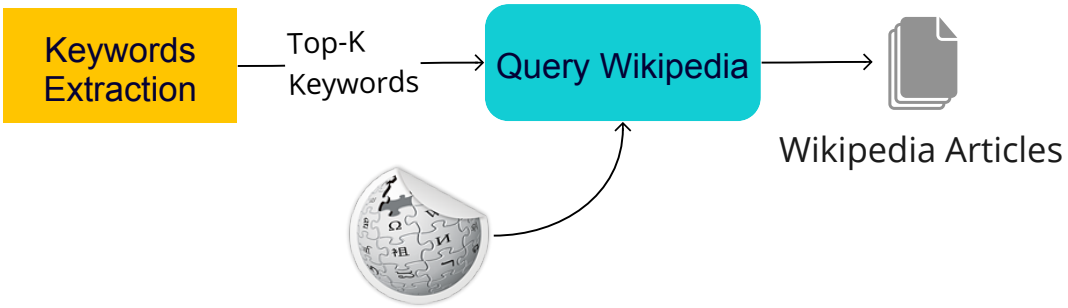


Figure 3.5: Domain-specific Corpus Generation.

discrete tags", corresponds to pattern#2 for PAA. Again, we recall from Section 3.1 that PAA occurs when it is unclear whether the PP in question is an adverbial modifier attached to **v** (*VA*) or a noun attribute attached to $\mathbf{n_1}$ (*NA*).

Step 2 identifies the segments (from the requirements in S_c and S_p) that match any of the patterns in Table 3.1. The matched segments are passed on to step 5.

3.3.3 Domain-specific Corpus Generation

This step attempts to capture the domain knowledge underlying the input requirements document (*SRS*) by crawling Wikipedia. Fig. 3.5 shows the sub-steps for generating a domain-specific corpus. We elaborate these (sub-)steps next.

Extract Keywords. Step 3.1 builds on an existing automated requirements glossary extraction approach by Arora et al. [101]. We begin by (automatically) extracting the list of glossary terms from *SRS*, and thereafter select the top-K most frequent keywords from the list. The optimal value of K is tuned in Section 3.4.4. For example, the keywords extracted from R1 (Fig. 3.1) include "LEO", "LEO satellites", "satellites", and "terminals". These keywords are used in the next step.

Query Wikipedia. Step 3.2 implements a query engine for identifying Wikipedia articles that are relevant to the keywords resulting from step 3.1. These articles form the basis of our domain-specific corpus. We begin by retrieving matching Wikipedia articles for each keyword. An article is considered a match if the keyword in question contains (or is contained in) the title of the Wikipedia article. For instance, the Wikipedia article titled "satellite navigation" is a match for the keyword "satellite-based navigation". If the above condition is not met, no matching Wikipedia article is retrieved.

Next, we broaden the domain information captured in our corpus by taking advantage of the hierarchical category structure of Wikipedia [94]. In Wikipedia's hierarchy, each category can contain articles and nested sub-categories. For a matching article, we retrieve all the articles in the same category and all the articles in the descendant sub-categories. For example, the "satellite navigation" article, as shown in Fig. 3.6, is classified under an identically named Wikipedia category (`https://en.wikipedia.org/wiki/Category:Satellite_navigation`; accessed 17/8/2020). We retrieve all articles in this category and in all its descendants (e.g., one descendant being "Geocaching"). Doing so increases topic coherence [102], meaning that the articles included in the corpus are all indeed relevant to the domain under analysis.

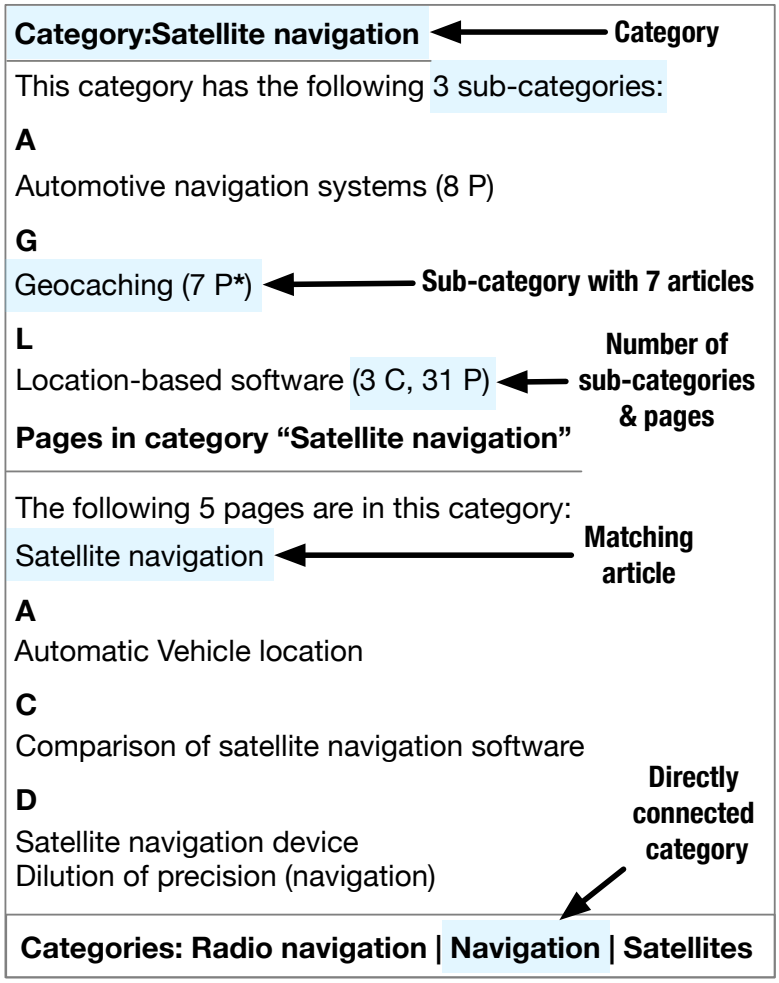


Figure 3.6: Example of Category Structure in Wikipedia.

Next, to make our domain-specific corpus applicable to other requirements documents from the same domain, we attempt to increase the coverage of our corpus. In particular, we consider the categories in the Wikipedia category graph that are directly connected to the category of the matching article. For instance, the category "navigation" in Fig. 3.6 is directly connected to "satellite navigation"; we therefore include articles listed under "navigation" and its descendant sub-categories.

We note that, during the creation of a corpus, we consider only the categories whose number of articles is below a threshold (α); this is both to keep the computation time reasonable and to avoid including large and generic categories in the corpus. We discuss the tuning of α in Section 3.4.4. The result of this step (step 3.2) is a body of raw text from Wikipedia articles. This extracted body of text is our domain-specific corpus, hereafter denoted as $\mathcal{D}$.

3.3.4 Application of Heuristics

Step 4 attempts to provide likely interpretations for the requirements in S_c and S_p. We use six heuristics, denoted as $\mathcal{C}_1$–$\mathcal{C}_6$, for interpreting coordination structures and four heuristics, denoted as $\mathcal{P}_1$–$\mathcal{P}_4$, for interpreting PP-attachment structures. Out of these ten heuristics, eight ($\mathcal{C}_1$–$\mathcal{C}_5$ for CA [68, 76, 77, 78] and $\mathcal{P}_1$–$\mathcal{P}_3$ for PAA [80, 79]) are borrowed from the literature. The other two ($\mathcal{C}_6$ and $\mathcal{P}_4$) are novel, but based on a very

intuitive idea: applying constituency parsing, which has coordination and PP-attachment interpretation built into it.

Similar to step 2 (Section 3.3.2), we operate at a *segment* level. Compared to the patterns in step 2, heuristics cover a wider spectrum of segments that have the potential to be ambiguous. The heuristics are triggered by the presence of any coordination structure (an "and" / "or" conjunction, two conjuncts and a modifier) and any PP-attachment structure (a verb, a noun and a PP). For example, had R2 (Fig. 3.2) contained an extra adjective "categorize outages with *standard* discrete tags", R2 would not have been detected by the patterns of Table 3.1, but would have been picked up by the heuristics and attempted for interpretation.

	Words	Count
Unigrams	system	360070
	satellite	21013
	navigation	11610
	orbit	26599
	...	
2-grams	navigation system	1284
	satellite orbit	234
	...	
3-grams	satellite navigation system	138
	low earth orbit	724
	...	
4-grams	global navigation satellite system	89
	geosynchronous satellite launch vehicle	27
	...	
5-grams	satellite power system concept development	8
	LEO sun synchronous receiver satellites	8

Figure 3.7: Excerpt of 5-grams Table.

Several heuristics in our approach are corpus-based, i.e., require information about the co-occurrence frequencies of the words. We therefore transform the Wikipedia articles from step 3 to an n-grams table with n ranging from 1 to 5. We set the upper limit to 5, motivated by the use of 5-grams in Google's well-known Web1T database [103]; this database is utilized in a wide variety of NLP applications [104, 105, 106].

Table 3.7 shows a (very small) excerpt of a 5-grams table generated for the satellite domain. The frequencies used by the heuristics are the normalized values of the co-occurrence counts listed in the 5-grams table [107]. For example, the co-occurrence frequency of "satellite orbit" is computed as $234/(21013 + 26599) \approx 0.005$.

Heuristics for CA. A segment in S_c contains a conjunction (c), two conjuncts (conjunct$_1$ and conjunct$_2$), and a modifier (mod). CA heuristics attempt to interpret a segment in S_c as either *FR* or *SR*. If a heuristic cannot interpret a segment, it returns a designated value, *not interpretable* (*NI*). As we explain below, four of the CA heuristics ($\mathcal{C}_1$ and $\mathcal{C}_3$–$\mathcal{C}_5$) require pre-defined thresholds, denoted as θ_i. These thresholds come from the existing literature. We empirically tuned the thresholds in Section 3.4.4. To illustrate the heuristics in this section, we already use the tuned θ_i values: $\theta_1 = 0.01$, $\theta_3 = 0.12$, $\theta_4 = 3.45$, and $\theta_5 = 3$.

($\mathcal{C}_1$) *Coordination frequency* computes the co-occurrence frequency of conjunct$_1$ and conjunct$_2$ in our domain-specific corpus ($\mathcal{D}$). We consider the co-occurrence frequency of the conjuncts irrespective of their order. For example, for R1, we consider, among other possible combinations, the co-occurrence frequency of "terminals and satellites" and "satellites or terminals". The intuition is that if the two conjuncts co-occur frequently in $\mathcal{D}$, they can be regarded as one syntactic unit and thus are both modified (by mod), in turn favoring *FR*. $\mathcal{C}_1$ returns *FR* if the resulting frequency is greater than a threshold (θ_1) and *NI* otherwise. In R1, $\mathcal{C}_1$ returns *FR*.

($\mathcal{C}_2$) *Collocation frequency* compares the co-occurrence frequency of conjunct$_1$ and mod against the frequency of conjunct$_2$ and mod. Collocation is a recurrent combination of two consecutive words in a large corpus [37]. For example, the words "red" and "wine" would be considered collocated, while "great" and "wine" would not. The intuition is that a collocation of the mod and the conjunct closer to it is likely to indicate a syntactic unit, thus favoring *SR*. Using collocations, "red wine and cheese" can be interpreted

as *SR* while "great wine and cheese" would not be interpretable (*NI*). $\mathcal{C}_2$ returns *SR* if the collocation frequency of mod and the closer conjunct is greater than that of mod and the farther conjunct, and *NI* otherwise. In R1, $\mathcal{C}_2$ returns *SR*.

($\mathcal{C}_3$) *Distributional similarity* measures the contextual similarity of conjunct$_1$ and conjunct$_2$ [108], i.e., how frequently the conjuncts appear in similar contexts. For example, in the context of requirements documents about satellite systems, the terms "satellite" and "navigation" have a higher distributional similarity than "satellite" and "investment". The intuition is that conjuncts with high distributional similarity can be regarded as one unit, thus favoring *FR*. $\mathcal{C}_3$ returns *FR* if the distributional similarity of the conjuncts is greater than θ_3, and *NI* otherwise. In R1, $\mathcal{C}_3$ returns *NI*.

($\mathcal{C}_4$) *Semantic similarity* measures the similarity between conjunct$_1$ and conjunct$_2$ based on their meanings in WordNet. The intuition is that conjuncts with high semantic similarity can be regarded as one unit, thus favoring *FR*. $\mathcal{C}_4$ returns *FR* if the semantic similarity is greater than θ_4, and *NI* otherwise. In R1, $\mathcal{C}_4$ returns *FR*.

($\mathcal{C}_5$) *Suffix matching* examines the number of shared trailing characters (suffixes) of conjunct$_1$ and conjunct$_2$. For example, "installation and configuration" share five trailing characters. Suffixes are used to change the meaning (e.g., "-able" in noticeable) or grammatical property (e.g., "-ed" in closed) of a given word [109]. Hence, matching suffixes provides a cue about how words are semantically or syntactically related [78]. The intuition is that conjuncts with the same number of trailing characters are likely to be a single unit, thus favoring *FR*. $\mathcal{C}_5$ returns *FR* if the conjuncts share trailing characters greater than θ_5, and *NI* otherwise. In R1, $\mathcal{C}_5$ returns *NI*.

($\mathcal{C}_6$) *Coordination syntactic* analysis is based on applying constituency parsing to the requirement in which the (coordination) segment of interest appears and then obtaining (from the parse tree) the interpretation of the parser for the segment. $\mathcal{C}_6$ returns *FR* or *SR* as per the parsing results, and *NI* if the parser fails to parse the requirement. In R1, $\mathcal{C}_6$ returns *FR*.

Heuristics for PAA. A segment in S_p contains a verb (v) and a following noun (n_1), followed by a preposition (p) and another noun (n_2). PAA heuristics attempt to interpret a segment as either *VA* or *NA*, as explained below. If a heuristic cannot interpret a segment, it returns *not interpretable* (*NI*).

($\mathcal{P}_1$) *Preposition co-occurrence frequency* compares the frequency of p occurring with v against p occurring with n_1. The intuition is that, based on the co-occurrence frequency of v (or n_1) and p, PP can be regarded as an adverbial modifier leading to *VA* or a noun attribute leading to *NA*. For example, in the segment "provide user with a valid option", the preposition "with" frequently follows the verb "provide", thus leading to a *VA* interpretation. Precisely, $\mathcal{P}_1$ returns *VA* if the co-occurrence frequency of v and p is strictly larger than that of n_1 and p. $\mathcal{P}_1$ returns *NA* if the converse is true. If there is a tie between the frequencies, e.g., when the frequencies are zero due to v, n_1 or p being absent from the corpus, $\mathcal{P}_1$ returns *NI*. In R2, $\mathcal{P}_1$ returns *NA*.

($\mathcal{P}_2$) *Prepositional-phrase (PP) co-occurrence frequency* has a similar definition and intuition to $\mathcal{P}_1$, the only difference being that we consider the entire PP (i.e., p and n_2) instead of just p. For example, consider the segment "provide [...]" used for illustrating $\mathcal{P}_1$. $\mathcal{P}_2$ would return *VA* because the PP "with a valid option"

has a higher co-occurrence frequency with v ("provide") than with n_1 ("user"). $\mathcal{P}_2$'s precise definition is easy to extrapolate from the definition of $\mathcal{P}_1$ and is omitted for space. In R2, $\mathcal{P}_2$ returns *NA*.

($\mathcal{P}_3$) *Semantic-class enrichment* utilizes the semantic classes in WordNet that group words with similar meanings. For example, WordNet puts "scissors" and "knife" under the same semantic class, namely "tool". $\mathcal{P}_3$ is applied after all the segments in S_p have been already processed by $\mathcal{P}_1$ and $\mathcal{P}_2$. Specifically, $\mathcal{P}_3$ attempts to find an interpretation for the segments that have been deemed as *NI* by both $\mathcal{P}_1$ and $\mathcal{P}_2$. For any such segment X, $\mathcal{P}_3$ checks whether there is some segment Y in S_p which has been interpreted as *VA* or *NA* (by either $\mathcal{P}_1$ or $\mathcal{P}_2$) and which shares a semantic class with X. By sharing a semantic class, we mean that X and Y contain nouns or verbs that fall under the same WordNet semantic class. If *Y* has been interpreted as *VA* (respectively, *NA*) and X shares a verb class (respectively, a noun class) with Y, then $\mathcal{P}_3$ interprets X as *VA* (respectively, *NA*).

The intuition is as follows: segments that contain words with similar meanings are likely to have the same interpretation [79]. For instance, a segment *X*: "offer operator with a valid option" is interpreted as *VA* by $\mathcal{P}_3$ if there is a segment *Y*: "provide user with a valid option" already interpreted as *VA* by $\mathcal{P}_2$. This is because the verbs "provide" and "offer" have the same WordNet semantic class: "possession".

($\mathcal{P}_4$) *Attachment syntactic analysis* has the same intuition and definition as $\mathcal{C}_6$, except that it applies to a PP-attachment segment. $\mathcal{P}_4$ returns *VA* or *NA*, as per the parsing results. $\mathcal{P}_4$ returns *NI* if the parser fails. In R2, $\mathcal{P}_2$ returns *VA*.

Combination of Heuristics. To produce a single interpretation for each segment, we combine through voting the results of the heuristics for each ambiguity type ($\mathcal{C}_1 - \mathcal{C}_6$ for CA and $\mathcal{P}_1$–$\mathcal{P}_4$ for PAA). We consider two voting methods: *majority voting* and *weighted voting* [110]. In majority voting, all heuristics contribute equally and the resulting interpretation is based on the majority. In weighted voting, the contribution of each heuristic is weighted differently. The weights are tuned in Section 3.4.4. In R1, majority voting yields *FR*, while weighted voting (using the tuned weights of Section 3.4.4) yields *SR*. We compare the accuracy of both voting methods in Section 3.4.

Step 4 partitions S_c and S_p into two subsets each: the first subset contains the *interpretable* segments (*FR* or *SR* for segments in S_c, and *VA* or *NA* for segments in S_p); the second subset contains the segments that are *not interpretable*. These subsets are passed on to step 5 for ambiguity handling.

3.3.5 Handling Ambiguity

In this final step, we classify into *ambiguous* and *unambiguous* the segments in S_c and S_p. This classification is based on the results of steps 2 and 4 in our approach (see Fig. 3.3). A segment X is classified as *ambiguous* if either of the following two conditions is met: (a) X matches some pattern in step 2, or (b) X is deemed as not interpretable (*NI*) in step 4. Any segment that is not classified as ambiguous would be *unambiguous*. We say that a requirement is ambiguous if it has some ambiguous segment; otherwise, we say the requirement is unambiguous. Our empirical evaluation, discussed next, is at a segment level (rather than a requirement level), because each requirement may contain multiple segments that are subject to ambiguity analysis.

3.4 Evaluation

In this section, we empirically evaluate our approach.

3.4.1 Research Questions (RQs)

Our evaluation addresses four research questions:

RQ1. What configuration of our approach yields the most accurate results for ambiguity handling? Our approach can be configured in a number of alternative ways; the alternatives arise from the choices available for the selection of patterns (Section 3.3.2), the use of default versus optimal thresholds for CA heuristics (Section 3.3.4), and the voting method for combining the heuristics (Section 3.3.4). RQ1 identifies the configuration that produces the best overall results.

RQ2. How effective is our approach at detecting unacknowledged ambiguity? As discussed in Section 3.1, unconscious misunderstandings may occur due to unacknowledged ambiguity. Using the best configuration from RQ1, RQ2 assesses the effectiveness of our approach in automatically detecting unacknowledged ambiguity.

RQ3. How accurate are the interpretations provided by our approach? While the exact interpretation of a segment found by our approach (*FR* or *SR* for segments in S_c, and *VA* or *NA* for segments in S_p) has no bearing on how we tell apart unambiguous cases from ambiguous ones, we want the interpretations to be as correct as possible. A correct interpretation is important both for reducing manual work (in case the analysts choose to vet the automatic interpretations), and also for ensuring that any subsequent automated analysis over the requirements, e.g., automated information extraction, will produce high-quality results. RQ3 examines the accuracy of the interpretations provided by our approach.

RQ4. Does our approach run in practical time? RQ4 studies whether the execution time of our approach is practical.

3.4.2 Implementation

We have implemented our approach (Fig. 3.3) in Java. The implementation has ≈8500 lines of code excluding comments. The NLP pipeline of step 1 is implemented using DKPro [111]. For implementing step 3, we use the English Wikipedia dump[1] timestamped 01/11/2018. We access the data in this dump using the JWPL library [112]. In step 4, we transform the raw text of Wikipedia articles into an n-grams table using the JWEB1T library [113]; this enables us to compute our interpretation heuristics more efficiently. We use Stanford Parser [114] to obtain the parse trees required by heuristics $\mathcal{C}_6$ and $\mathcal{P}_4$. For $\mathcal{C}_4$, we compute semantic similarity using the Resnik measure [115] as implemented by the WS4J library [116]. For implementation availability, please see the footnote on page 2.

3.4.3 Data Collection

Our data collection involves human experts studying and annotating potential CA and PAA in industrial requirements. We collected our data from 20 requirements documents (*SRS* s) written in English and originating from three distinct industry partners. These SRSs cover seven different application domains. Data collection was performed by two third-party annotators (nonauthors) with expertise in linguistics. The first annotator, Anna (pseudonym), has a Master's degree in Multilingualism. Anna had previously completed a three-month

[1] `https://dumps.wikimedia.org/backup-index.html`

	Domain	Aerospace	Automative	Defense	Digitalization	Medicine	Satellite	Security	Total
	SRSs	5	3	2	3	1	4	2	20
	Total Requirements	1510	284	910	701	189	1420	142	5156
S_c	Requirements	295	49	294	164	28	265	3	1098
	Segments	382	64	416	250	39	352	3	1506
	Acknowledged	78	21	164	36	0	70	1	370
	Unacknowledged	171	15	19	109	19	154	0	487
	Unambiguous	133	28	233	105	20	128	2	649
S_p	Requirements	353	44	238	272	19	333	69	1328
	Segments	388	47	266	312	20	360	81	1474
	Acknowledged	117	8	29	105	5	14	0	278
	Unacknowledged	115	17	131	77	5	174	36	555
	Unambiguous	156	22	106	130	10	172	45	641

Table 3.2: Data Collection Results.

internship in RE. The second annotator, Nora (pseudonym), has a Masters degree in IT Quality Management. Nora has a professional certificate in English translation. Both annotators underwent a half-day training on ambiguity in RE. The two annotators produced their annotations over a 6-month period, during which they declared a total of ≈130 and ≈165 hours, respectively.

The annotators were then tasked with independently labeling with *FR* or *SR* all the ("and"/"or") coordination segments in S_c and labeling with *VA* or *NA* all the PP-attachment segments in S_p. The annotators were specifically instructed to ascribe an interpretation to a segment only when they were confident about their interpretation. When in doubt, the annotators labeled the segment in question as ambiguous. An "agreement" between annotators is observed for segment X, when both of them either find X ambiguous or interpret X the same way. Any other situation is a "disagreement". Using Cohen's kappa metric (κ) [117], we obtain an inter-rater agreement of 0.37, suggesting "fair agreement". To examine the sources of disagreement, we further analyze the cases where X is deemed ambiguous (i.e., acknowledged ambiguity). For these cases, we obtain $\kappa = 0.78$ ("substantial agreement"), indicating that most disagreements are due to different interpretations (i.e., unacknowledged ambiguity). As stated earlier in the chapter, unacknowledged ambiguity is believed to be prevalent in requirements [19, 68]. The analysis, discussed above, provides empirical evidence for this belief.

We constructed our ground truth as follows: (i) any segment labeled as ambiguous by at least one annotator is a case of *acknowledged ambiguity*, (ii) any segment labeled with different interpretations by the annotators is a case of *unacknowledged ambiguity*, and (iii) any segment labeled with the same interpretation by both annotators is *unambiguous*. We motivate our definitions of acknowledged and unacknowledged ambiguity by considering what might happen during a manual inspection where a team would typically be involved. If a segment is ambiguous enough for someone (not necessarily everyone) to raise a concern, then this segment is likely to be further discussed by the team (acknowledged). The situation is different for unacknowledged ambiguity. In reality and under time pressure, the analysts are unlikely to spell out their interpretations when they feel there is no ambiguity. Consequently, the disagreement remains hidden (unacknowledged).

Table 3.2 provides overall statistics about our data collection, showing for each domain, the number of

*SRS*s, the total number of requirements, and the number of requirements and segments in S_c and S_p. The table further lists the number of ambiguous segments (grouped into acknowledged and unacknowledged) and the number of unambiguous segments. We observe from Table 3.2 that out of the total of 2980 segments analyzed by the annotators, 57% are ambiguous and the remaining 43% are unambiguous. In the ambiguous segments, the proportion of segments with unacknowledged ambiguity (1042/1690 $\approx$ 62%) is significantly higher than the proportion of segments with acknowledged ambiguity (648/1690 $\approx$ 38%). We note that repeated segments constitute a relatively small fraction of the ground truth: $\approx$9% (137/1506) for CA and $\approx$8% (116/1474) for PAA. These repetitions are not disproportionately concentrated in one group. More precisely, in the case of CA, 44 repetitions are unambiguous, 48 are acknowledged, and 45 are unacknowledged. For PAA, 39 repetitions are unambiguous, 35 are acknowledged, and 42 are unacknowledged. Since there is no disproportionate concentration of occurrences, repetitions have no major bearing on our findings.

We set aside $\approx$20% of our ground truth for parameter tuning, as we will discuss in Section 3.4.4. We refer to this subset of the ground truth as T. The remaining $\approx$80% of the ground truth is referred to as E. We use E for answering all the RQs, except RQ4 which is answered over $T \cup E$. The tuning set, T, consists of six *SRS*s from six domains with a total of 550 requirements and representing 26% and 21% of the coordination and PP-attachment segments, respectively. We selected one *SRS* from each domain; this was done in a way that the selected document would be as close as possible to containing $\approx$20% of the requirements we had in each domain. We did not select for tuning any documents from the domain of medicine, since we had only one *SRS* from this domain.

3.4.4 Parameter Tuning

Tuning involves two groups of parameters: parameters for generating a domain-specific corpus (Section 3.3.3) and parameters associated with the heuristics (Section 3.3.4). Both groups of parameters are tuned with the goal of maximizing the overall accuracy of the interpretation heuristics. Note that tuning is performed exclusively over T (see Section 3.4.3).

Parameters for Corpus Generation. Generating a domain-specific corpus requires tuning the maximum number of keywords (K) to select from an input *SRS* and the maximum number of articles (α) in a given category in Wikipedia. For each *SRS* in T, we generate a domain-specific corpus. To tune K, we experiment with five values at regular intervals between 50–250. Values of K outside this range are undesirable as they result in a corpus that is either too small (for $K < 50$) or too large (for $K > 250$). A suitably large corpus is necessary for accurately estimating the co-occurrence frequencies of words in a specific domain [107]. Building and using a corpus that is too large would be time-consuming and, more importantly, would defeat the goal of being domain-specific. Using a corpus that is too small would simply be ineffective. For tuning α, we experiment with values in the range of 50–1000 in intervals of 50. Larger categories (i.e., $\alpha > 1000$) are too generic, and smaller ones (with $\alpha < 50$) are already covered by $\alpha > 50$, as α is the upper bound for the number of articles in a category. For optimizing K and α, we use grid search [118]. The resulting optimal values are $K = 100$ and $\alpha = 250$.

Parameters for Heuristics. Applying the interpretation heuristics requires tuning four thresholds θ_1, θ_3–θ_5 respectively for heuristics $\mathcal{C}_1$, $\mathcal{C}_3$–$\mathcal{C}_5$. For using the weighted voting method, we further need to tune the weights of all the heuristics.

We note that the thresholds for the heuristics have been introduced and tuned in the existing literature, albeit for generic texts [76, 68, 77]. We re-tune these thresholds to better capture co-occurrence frequencies in the context of requirements. The threshold values from the existing literature are hereafter referred to as *default*.

We experiment with 1000 regular intervals in the range of 0.01–10 for tuning θ_1, θ_3 and θ_4. To tune θ_5, we investigate suffixes of lengths 1 to 5, e.g., the suffix "-ation" has a length of five. We use random search [118] to optimize the thresholds because the search space is too large for grid search. The resulting *optimal* thresholds are $\theta_1 = 0.01$, $\theta_3 = 0.12$, $\theta_4 = 3.45$, and $\theta_5 = 3$.

For determining the weights of the heuristics, we first apply each heuristic individually on T. The weight of a given heuristic is determined by its success in providing interpretations for the segments. In our experiments, the weights of heuristics in descending order for CA are 0.038 for $\mathcal{C}_5$, 0.019 for $\mathcal{C}_2$, 0.012 for $\mathcal{C}_1$, 0.005 for $\mathcal{C}_4$, 0.005 for $\mathcal{C}_6$ and 0.003 for $\mathcal{C}_3$, and the weights for PAA are 0.08 for $\mathcal{P}_1$, 0.05 for $\mathcal{P}_2$ and 0.03 for $\mathcal{P}_4$. $\mathcal{P}_3$ is not a standalone heuristic and is thus not weighted. These weights reflect the contribution of the heuristics, in weighted voting, to produce a final interpretation for a segment.

3.4.5 Evaluation Procedure

We answer our RQs through the following experiments.

EXPI. This experiment answers **RQ1**. We determine the optimal configuration for ambiguity handling by comparing the output of our approach against E. For evaluating the configurations, we define a *true positive (TP)* as a detected ambiguous segment, a *true negative (TN)* as an unambiguous segment marked as such, a *false positive (FP)* as a misclassified unambiguous segment, and a *false negative (FN)* as a misclassified ambiguous segment. We compute *Accuracy* (A) as $(TP + TN)/(TP + TN + FP + FN)$, *Precision* ($P$) as $TP/(TP + FP)$, and *Recall* (R) as $TP/(TP + FN)$.

We consider eight alternative configurations for our approach, denoted as $\texttt{V}_1$–$\texttt{V}_8$. These alternatives are induced by three binary decisions. The first decision is whether to use the *collected* or the *enhanced* patterns in step 2 of our approach (see Table 3.1). The second decision is whether in step 4 we should use for the thresholds the *default* or the *optimal* values (from Section 3.4.4). And, the third decision is whether the method for combining the heuristics is *majority* or *weighted* voting (see Section 3.3.4). To analyze the impact of using domain-specific corpora, we compare our approach against baselines, denoted as $\texttt{B}_1$–$\texttt{B}_8$, with similar configurations but using a generic corpus: the British National Corpus [119].

To run EXPI, we first need to generate seven corpora, one for each application domain in our ground truth (see Table 3.2). Six of these corpora are reused from Section 3.4.4. The last one – for the domain of medicine – is generated based on the single *SRS* we have in our dataset for this domain. Except for the domain of medicine, EXPI provides an implicit assessment of how reusable a domain-specific corpus is, being generated from one *SRS* and reused for other *SRS*s from the same domain.

EXPII. This experiment answers **RQ2**. Given the optimal configuration of our approach from EXPI, EXPII assesses how well our approach can detect unacknowledged ambiguity in different domains. In EXPII, we compute *Recall (R)* similar to EXPI, but limiting the evaluation to only the segments with unacknowledged ambiguity in E. The corpora used in EXPII are the same as those in EXPI.

EXPIII. This experiment answers **RQ3**. We evaluate the interpretations provided by our approach for the segments classified as unambiguous (*FR* or *SR* for segments in S_c, and *VA* or *NA* for segments in S_p). Specifically, EXPIII compares the interpretations produced by our approach against the interpretations of unambiguous segments in E, reporting the ratio of the correctly interpreted segments (i.e., *Accuracy*). The corpora used in EXPIII are the same as those in EXPI and EXPII. We further compare our approach against Stanford Parser [114] – one of the commonly used tools for interpreting syntactic ambiguity [120].

					CA			PAA		
		Patterns	Thresholds	Voting	A (%)	P (%)	R (%)	A (%)	P (%)	R (%)
V_1	Domain-Specific Corpus	collected	default	majority	70.8	76.9	66.6	78.4	77.8	84.0
V_2		collected	default	weighted	71.6	78.0	66.9	78.9	78.5	84.0
V_3		collected	optimal	majority	74.5	81.1	69.5	82.0	81.5	86.4
V_4		collected	optimal	weighted	75.3	82.2	69.9	82.5	82.2	86.4
V_5		enhanced	default	majority	76.7	75.6	84.6	78.6	76.3	87.7
V_6		enhanced	default	weighted	77.5	76.4	84.9	79.1	76.9	87.7
V_7		enhanced	optimal	majority	80.5	78.9	87.6	82.2	79.8	90.1
V_8		enhanced	optimal	weighted	81.3	79.8	87.9	82.7	80.3	90.1
B_1	British National Corpus	collected	default	majority	37.6	42.5	40.6	45.7	49.9	53.5
B_2		collected	default	weighted	38.4	43.2	40.9	46.2	50.2	53.5
B_3		collected	optimal	majority	41.3	46.0	43.6	49.3	53.1	55.9
B_4		collected	optimal	weighted	42.1	46.8	43.9	49.8	53.5	55.9
B_5		enhanced	default	majority	42.9	48.2	60.4	45.8	50.0	57.2
B_6		enhanced	default	weighted	43.7	48.8	60.8	46.2	50.3	57.2
B_7		enhanced	optimal	majority	46.7	50.9	63.3	49.4	52.9	59.6
B_8		enhanced	optimal	weighted	47.5	51.6	63.8	49.9	53.3	59.6

Accuracy (A), Precision (P) and Recall (R) in percentage (%)

Table 3.3: Results of Ambiguity Handling (RQ1).

EXPIV. This experiment answers **RQ4** by running the best configuration from RQ1 over $T \cup E$. The experiment is done on a laptop with a 2.3 GHz CPU and 16GB of memory.

3.4.6 Answers to the RQs

RQ1. Table 3.3 shows the results of EXPI (on E). To determine the optimal configuration of our approach, we investigate among all configurations the factors that cause the most variation in accuracy. We do so by performing regression tree analysis (tree not shown) [121]. The most influential factor for both CA and PAA, as per regression tree analysis, is the choice of domain-specific versus generic corpus. *The configurations that use a domain-specific corpus,* V_1–V_8*, have an average gain in accuracy of ≈33% over the configurations that use a generic corpus,* B_1–B_8*. This observation clearly highlights the importance of domain knowledge in ambiguity handling.*

Among V_1–V_8, using enhanced patterns has a considerable impact on detecting CA. Compared to the configurations with collected patterns (V_1–V_4), the configurations with enhanced patterns (V_5–V_8) lead to an average gain of ≈6% in accuracy and ≈18% in recall for a minor ≈2% drop in precision. Compared to collected patterns, enhanced patterns do not improve the detection of PAA, but do not perform any worse either. Thus, we choose the enhanced patterns over the collected ones.

With respect to the thresholds for the heuristics, the configurations with optimal thresholds (V_7–V_8) outper-

	Domain	Aerospace	Automative	Defense	Digitalization	Medicine	Satellite	Security	Summary
CA	TP	145	13	17	28	19	46	0	268
	FN	25	2	2	4	0	6	0	39
	R (%)	84.7	86.6	89.4	87.5	100	88.4	-	87.3
PAA	TP	83	15	120	17	4	141	32	412
	FN	6	2	11	1	1	12	4	37
	R (%)	93.2	88.2	91.6	94.4	80.0	92.1	88.8	91.8

TP, FN: number of true positives and false negatives, R: Recall in percentage (%)

Table 3.4: Unacknowledged Ambiguity Detection using $\mathtt{V}_8$ (RQ2).

form those with default thresholds ($\mathtt{V}_5$–$\mathtt{V}_6$) by 3.7% in terms of accuracy. Noting that our parameter tuning used documents from six different application domains, we believe that the optimal thresholds are more suitable in an RE context than the default ones based on generic texts. We note that, overall, the accuracy of ambiguity handling shows little sensitivity to the choice of voting method. However, as highlighted in Table 3.3, $\mathtt{V}_8$ (weighted voting) is slightly more accurate than $\mathtt{V}_7$ (majority voting). For the subsequent RQs, *we select* $\mathtt{V}_8$ *as the best-performing configuration of our approach with enhanced patterns, optimal thresholds and weighted voting.*

To be able to perform a thorough error analysis (Section 3.4.7), we run $\mathtt{V}_8$ on the entire dataset ($T \cup E$). This yields a precision and recall of 80.1% and 89.3% for CA, and 81.6% and 90.2% for PAA, respectively. We observe that, for each metric, the overall results are only marginally (≈1%) better than what was reported over E. This provides confidence that our tuning (Section 3.4.4) did not overfit.

RQ2. The results of EXPII, obtained from running $\mathtt{V}_8$ (the best configuration from RQ1) *on* E are shown in Table 3.4. Overall, *our approach detects unacknowledged ambiguity with an average recall of 87.3% for CA and 91.8% for PAA.*

Our error analysis (Section 3.4.7) examines missed cases of unacknowledged ambiguity in the entire dataset ($T \cup E$). Over the entire dataset, $\mathtt{V}_8$ detects unacknowledged ambiguity with an average recall of 87.8% for CA and 92.6% for PAA.

RQ3. The interpretations provided by $\mathtt{V}_8$ for the segments in S_c and S_p (when restricted to E) have an average accuracy of 85.2% and 84.4%, respectively. The accuracy of the approach on the entire dataset is marginally higher (by an average of ≈1%). We examine interpretations errors in Section 3.4.7.

Applying the Stanford Parser to S_c and S_p (when restricted to E) yields interpretations with an average accuracy of 65.7% and 72.6%, respectively. *In an RE context and in comparison to the Stanford Parser, the integration of domain knowledge increases the interpretation accuracy of coordination and PP-attachment structures by an average of ≈16%.*

RQ4. Executing steps 1 and 2 of our approach (Fig. 3.3) takes ≈0.2 milliseconds per requirement. Step 3 is performed only when a suitable corpus is absent, i.e., when no corpus has been generated before for the domain of a given *SRS*, or when the domain of the *SRS* is difficult to ascertain. Across the seven corpora we generated for answering RQ1-3, the average execution time was ≈58 minutes (standard deviation: ≈21 minutes). To be able to generate corpora, there is a one-time overhead of ≈3 hours; this is to set up a query engine over

Wikipedia (see step 3.2 in Section 3.3.3). Once set up, this query engine does not have to be rebuilt, unless one wants to switch to a different edition of Wikipedia. With a corpus at hand, execution time is dominated by the computation of the frequencies required by the heuristics of step 4. This on average takes ≈6.8 seconds for a requirement in S_c and ≈1.5 seconds for one in S_p. Processing the requirements in S_c takes longer because there are more corpus-based heuristics for CA than PAA. Non-corpus-based heuristics take negligible time.

Excluding corpus generation, the largest document in our dataset took ≈51 minutes to process. This document had 492 requirements with 392 coordination and 245 PP-attachment segments. *Such an execution time is practical for offline (e.g., overnight) processing.* With regard to using our approach interactively, we observe that, at any point in time, an analyst likely works on only a small part of a large document. *For interactive use, ambiguity handling can be localized to the document fraction (e.g., page) that the analyst is reviewing.*

3.4.7 Error Analysis

In this section, we analyze the root causes of the errors made by our approach ($\mathtt{V}_8$) on the entire dataset ($T \cup E$).

Errors in RQ1 and RQ2. Out of 1690 segments (Table 3.2), our approach missed 192 ambiguous segments, of which 100 are unacknowledged. These errors can be explained as follows.

1. *Coverage of patterns: 169 segments* do not match any pattern in Table 3.1. For example, the segment "register the microservice in the operations server" matches no PAA pattern. One can avoid such errors by expanding the pattern set. However, our experiments indicate that doing so comes at the cost of a large number of FPs and is thus not worthwhile.

2. *NLP errors: 23 segments* are missed due to mistakes by the NLP pipeline (Fig. 3.4). For example, "support" in the segment "[can] support doctors in the ICU" is erroneously tagged as a noun; this results in the segment to not match any of our patterns. Such NLP mistakes are hard to avoid [122].

Errors in RQ3. We found two causes for interpretation errors.

1. *Interpretation errors by the heuristics: 74 segments* fall under this class of errors, having to do with situations where the combination of heuristics provide a wrong interpretation or return *not interpretable (NI)* where there is indeed an interpretation. For example, for the segment "pulse width and duration" the resulting interpretation is *SR*, although it should be *FR*. One can try to address individual interpretation errors by adjusting the weights of the heuristics. However, doing so will have a negative overall impact by causing other errors.

2. *Document-specific abbreviations: 58 segments* are misinterpreted due to abbreviations. An abbreviation that is specific to a document can mislead frequency computations if the abbreviation has a homonym or is not found in the corpus at all. For example, "MOC" in "MOC operator and component" stands for "MOnitoring and Control" in one of our *SRS*s from the satellite domain. This abbreviation, however, matches "Mars Orbiter Camera" in the corpus that we generate for this domain. Such mismatches can be reduced through abbreviation disambiguation [123]. We leave this for future work.

3.4.8 Discussion about Usefulness

As shown by Table 3.2, ambiguity was acknowledged by the annotators in only 38% of the cases. The remaining 62% were unacknowledged. In practice, even if the analysts perform a manual review, under time pressure and

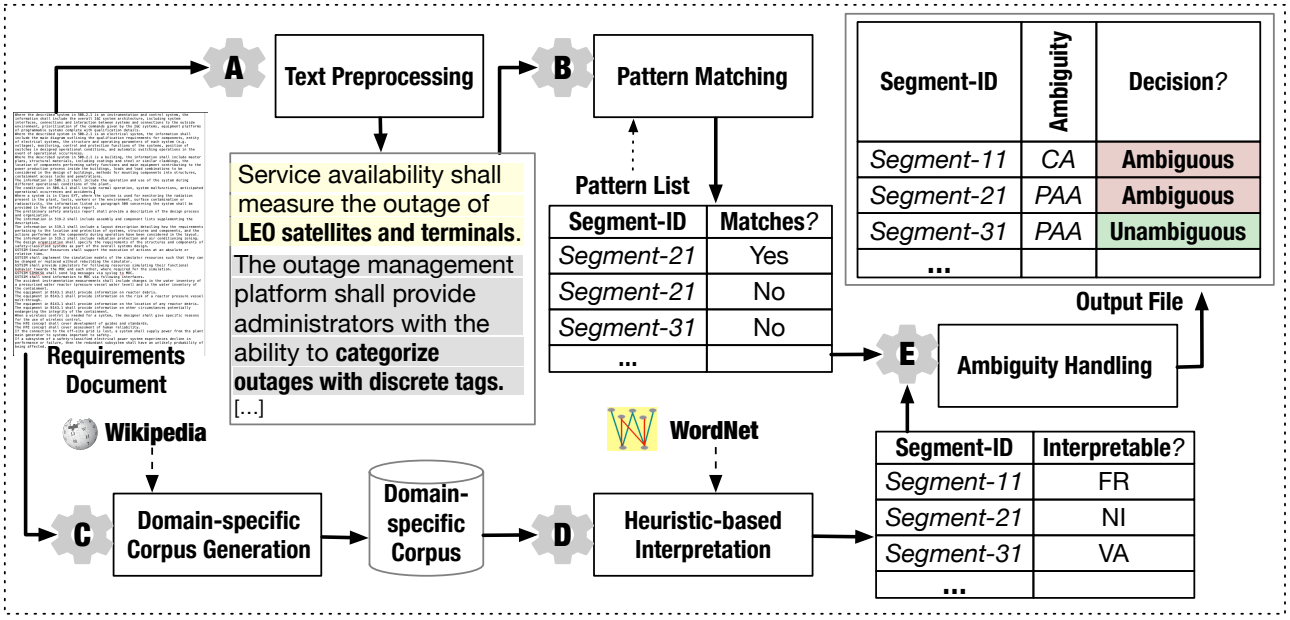


Figure 3.8: Tool Architecture.

outside an evaluation setting, they will likely only examine what at least one analyst finds to be ambiguous and thus miss out on the cases where they unconsciously disagree about the interpretation.

We believe that the main benefit of our automated approach is in bringing ≈90% of the cases of unacknowledged ambiguity to the attention of the analysts (see RQ2). This is achieved while maintaining a high overall precision (≈80%), meaning that the analysts will spend a small fraction of their manual effort over false positives. While user studies remain essential for establishing usefulness, our good accuracy results suggest that our approach has the potential to be helpful in practice.

3.5 Tool Support

This section presents the tool support for our ambiguity handling approach.

3.5.1 MAANA

MAANA is the implementation of our approach for detecting coordination ambiguity (CA) and prepositional-phrase attachment ambiguity (PAA).

Fig. 3.8 depicts the architecture of MAANA. The tool is implemented as an Apache Maven project [124] and builds on the Apache UIMA framework [125]. Below, we discuss the main modules of MAANA, marked A – E in Fig. 3.8.

Text Preprocessing

This module parses the text of the input requirements document, splitting it into sentences and identifying the requirements that contain coordination and/or PP-attachment segments. The NLP pipeline that MAANA applies consists of a tokenizer, sentence splitter, part-of-speech (POS) tagger, lemmatizer, and constituency parser.

Pattern Matching

This module matches the coordination and PP-attachment segments in the input document against a predefined set of patterns. These patterns are meant at identifying those segments that have a higher chance of being ambiguous. The output for each segment is whether or not it matches any of the patterns in the list.

Domain-specific Corpus Generation

This module generates a domain-specific corpus from Wikipedia based on (automatically identified) keywords in the input requirements document. For further information about how to set up Wikipedia in order to generate a corpus for a given input document, consult the "Additional Instructions" provided alongside our actual artifact.

Heuristic-based Interpretation

This module subjects the identified coordination and PP-attachment segments to a set of interpretation heuristics, as described in Section 3.3. The output is a combined decision based on these heuristics. NI suggests that a segment is not interpretable. For coordination, FR and SR suggest that a segment should be interpreted as first read or second read, respectively. And for PP attachment, VA and NA suggest that a segment should be interpreted as a verb attachment or a noun attachment, respectively.

Ambiguity Handling

The final module combines the results of modules (B) and (D) to obtain a final verdict for each identified segment. This final verdict is written out to the output file.

3.5.2 WikiDoMiner

WikiDoMiner (**Wiki**pedia **Do**main-specific **Miner**) is an improved and optimized tool for the domain-specific corpus generation component. Given an SRS as input, WikiDoMiner automatically generates a domain-specific corpus from Wikepedia, without any a-priori assumptions about the domain of the input SRS. WikiDoMiner is a re-implementation of the corpus generator in an earlier research prototype, MAANA [10]. MAANA is an automated ambiguity handling tool which uses frequency-based heuristics to detect coordination and prepositional-attachment ambiguity. In that context, a large domain-specific corpus is needed for estimating word frequencies. In our ongoing research since MAANA, we have increasingly needed domain-specific corpus generation, not for frequency-based statistics but rather for fine-tuning pre-trained language models. This prompted us to build and release WikiDoMiner as a stand-alone tool and a more robust and usable alternative to the corpus generator in MAANA. MAANA's corpus generator is Java-based. Furthermore, it requires a local dump of Wikipedia installed as an SQL database. This consumes significant resources and makes both the installation and (re-)use of MAANA complex. WikiDoMiner lifts this major limitation and further, by virtue of being Python-based, is much easier to use alongside language models.

WikiDoMiner is a usable prototype tool for generating domain-specific corpora. Figure 3.9 shows an overview of WikiDoMiner architecture. The tool is implemented in Python 3.7.13 [126] using Google Colab[2]. Below, we discuss the different steps of the tool marked A – C in Figure 3.9.

[2] `https://colab.research.google.com/?utm_source=scs-index`

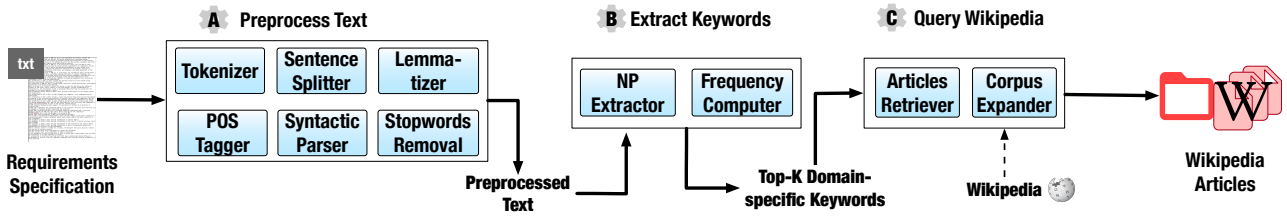


Figure 3.9: Tool Architecture.

Preprocess Text

In the first step, we parse the textual content of the input SRS and preprocess the text. To do so, we apply an NLP pipeline composed of six modules, four of which are related to parsing and normalizing the text, and two are for performing syntactic parsing. These modules include: A *tokenizer* splits the text into different tokens (e.g., commas and words), *sentence spitter* identifies the boundaries of sentences in the running text (e.g., a sentence in English can end with a period), *lemmatizer* finds the canonical form of a word (e.g., the singular word "communication" is the canonical form of its plural variant "communications" and the infinitive "transmit" is the canonical form for its past-tense variant "transmitted"), and finally, a *stopwords removal* module removes the stopwords such as articles ("the") and prepositions (e.g., "in"). To perform syntactic analysis, we further apply: *POS tagger* that assigns a part-of-speech tag for each token (e.g., the tag VBD is assigned to "transmitted" indicating a past-tense verb), and a *syntactic parser* that identifies the syntactic units in the text (e.g., "the notification service" is a noun phrase – NP).

To operationalize the NLP pipeline, we use the Tokenizer, Porter Stemmer and WordNet Lemmatizer available in NLTK 3.2.5 [127]. We further apply Python RE 2.2.1 regex library[3], in addition to available modules in SpaCy 3.3.0 [128] including the English stopwords list, Tokenizer, NP Chunker, Dependency Parser, and Entity Recognizer.

Extract Keywords

In this step, we extract a set of keywords that are representative for the underlying domain. To do that, we adapt a glossary extraction method from the RE literature [101]. The basic idea in this step is to collect the noun phrases in the input SRS, and sort them according to their frequency of use. To ensure that these keywords are domain-specific, WikiDoMiner applies two measures. First, we remove from the list any keyword that is available in WordNet [98, 99], which is a generic lexical database for English. The intuition of this step is to remove very common words that are not representative of the underlying domain. For instance, the word "rover" exists in WordNet[4] as a noun referring to "someone who leads a wandering unsettled life" or "an adult member of the Boy Scouts movement". These two meanings do not fit the "rover" in the "lunar rover" context, and the NP "lunar rover". This way, we filter out the word "rover" when it occurs alone (i.e., "the rover"), and keep it as part of the NP ("lunar rover"). We note that the the NP "lunar rover" is not available in WordNet, but is in Wikipedia[5].

As a second measure, WikiDoMiner computes term frequency/inverse document frequency (TF/IDF) [129] instead of mere frequency. TF/IDF is a traditional method that is often applied in the context of information

[3] https://docs.python.org/3/library/re.html

[4] http://wordnetweb.princeton.edu/perl/webwn?s=rover&sub=Search+WordNet&o2=&o0=1&o8=1&o1=1&o7=&o5=&o9=&o6=&o3=&o4=&h=

[5] https://en.wikipedia.org/wiki/Lunar_rover

retrieval (IR) to assign a score reflecting the importance of words to a specific document in a document collection. In WikiDoMiner, the importance of the words (NPs in our case) indicates that the words are significant for the underlying domain. We note that IDF is computed only when there are multiple documents from other domains available. Otherwise the TF/IDF scores are similar to term frequencies. Once the TF/IDF scores are computed, we sort the keywords in descending order of these scores and select the top-K keywords. While the default value applied by WikiDoMiner is $K = 50$, we show in the demo of the tool that this parameter can be configured by the user according to the intended application.

We implement the different modules using WordNet from NLTK 3.2.5 [127], and TF-IDF transformation from Scikit-learn 1.0.2 [130].

Query Wikipedia

In this step, we use the keywords from the previous set to query Wikipedia and collect the relevant articles which will then constitute our final domain-specific corpus.

To better understand this step, we first explain the structure of a category in Wikipedia, illustrated in Figure 3.10. Each article in Wikipedia belongs to one or more categories. Each category contains a set of articles and sub-categories. To illustrate, assume that we are querying Wikipedia for the keyword "rail transport" within the "Railway" domain. Our first hit will be a page titled "Rail Transport"[6]. Note that we refer to a page in Wikipedia as an *article*. If we view the category structure for this article[7], we find out that it belongs to a category under the same name "Rail Transport", i.e., Category A in Figure 3.10. Inside this category, there are 31 sub-categories such as "Locomotives", "Rail Infrastructure", and "Electric rail transport". Category A contains 22 other pages alongside the above mentioned pages, such as "Bi-directional vehicle" and "Pocket wagon". Viewing the structure of a sub-category, e.g., "Rail Infrastructure" will show us again the available pages and sub-categories.

In WikiDoMiner, the result of querying Wikipedia for a given keyword is a Wikipedia article whose title partially matches the keyword. We consider the title of a Wikipedia article as partially matching the keyword if they have some overlap. For example, if we query Wikipedia for the keyword "Efficiency of rail transport", then we will retrieve the same article mentioned above whose title, "Rail Transport", partially matches the keyword.

For each keyword, we retrieve from Wikipedia a matching article if applicable. Some applications might require that the domain-specific corpus be sufficiently large. For example, to accurately estimate the frequencies of words co-occurrences, one needs a large corpus [131]. Similarly, to pre-train a domain-specific language model, a large text body should be available. Therefore, we expand our corpus by defining a configurable parameter *depth* to control the level of expansion, thus allowing the user to adjust the size and relevance of the corpus based on their needs. The minimal depth $depth = 0$ can be used to extract directly matching articles only (leading most often to a few hundred articles). WikiDoMiner further retrieves, for each matching article, all articles in the same categories for $depth = 1$ (e.g., the two other pages in the example above), subcategories of $depth = 2$, sub-subcategories of $depth = 3$, and so on.

Specific details of our implementation are as follows. We use the Wikipedia 1.4.0[8] and Wikipedia-API 0.5.4[9] libraries to query Wikipedia. Other libraries which we use but which are not necessary to run the tool include

[6] https://en.wikipedia.org/wiki/Rail_transport
[7] https://en.wikipedia.org/wiki/Category:Rail_transport
[8] https://wikipedia.readthedocs.io/
[9] https://wikipedia-api.readthedocs.io/

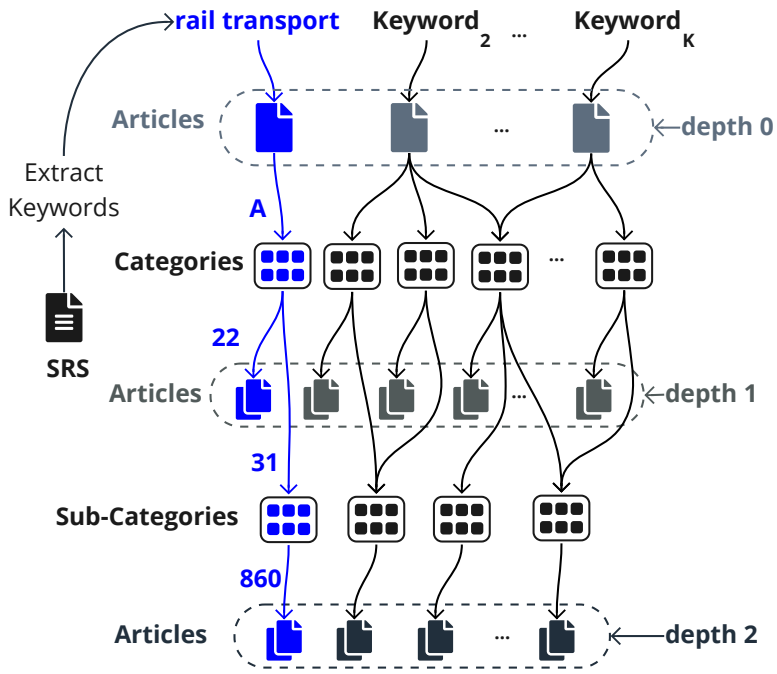


Figure 3.10: Illustration of Traversing Wikipedia Categories (Example Keyword: "rail transport").

PyPDF2 2.2.0[10] to read requirements documents in PDF format, the word2vec similarity feature in SpaCy 3.3.0 library [128], and the WordCloud 1.5.0[11] library to visualize the most prevalent words in the extracted corpora.

Application

In this subsection, we apply WikiDoMiner to automatically generate domain-specific corpora for two distinct domains, namely, railway and transportation. We further assess how representative the corpus generated for each of these domains is. We do so by computing the semantic relatedness of each domain-specific corpus against SRSs from the same domain other than those used for generating the corpus. Generating a domain-specific corpus is not a frequent activity. In practice, requirements engineers would typically have a small set of SRSs from a given domain at the time of generating a domain-specific corpus and would utilize this corpus to perform activities on other SRSs not involved in the generation process.

Data Collection For the two domains considered in this subsection, we collected a total of six SRSs from the PURE dataset [132], with three SRSs from each domain. One SRS is used for generating the corpus and the others are used for evaluating semantic relatedness against the resulting corpus.

In the following we list the six SRSs:

- From the railway domain, we used SRS1 (*ERTMS*) about *train control*, SRS2 (*EIRENE SYS 15*) and SRS3 (*EIRENE FUN 7*) both about *digital radio standard for railway*.
- From the transportation domain, we used SRS4 (*CTC NETWORK*) about *traffic management infrastructure*, SRS5 (*PONTIS*) about *highway bridge information management*, and SRS6 (*MDOT*) about *transportation info management*.

[10] https://pypdf2.readthedocs.io/
[11] https://amueller.github.io/word_cloud/

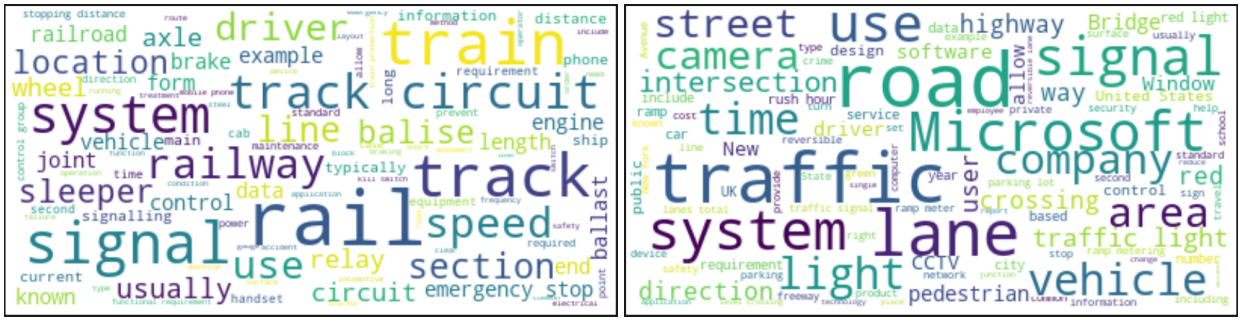


Figure 3.11: Word-cloud Visualization of Domain-specific Corpora (Left-hand Side – Railway Domain, and Right-hand Side – Transportation Domain).

Domain-specific Corpora For illustration, we centre our discussion around the railway domain. We generate the corpus from SRS1, and evaluate the relatedness on SRS2 and SRS3. The first step in WikiDoMiner is to preprocess SRS1. WikiDoMiner then extracts a set of keywords based on their TF/IDF scores. Examples of such keywords include *trainborne equipment* and *emergency brake*. We select the top-K keywords, where $K = 50$.

The next step is to query the keywords on Wikipedia. For our set of keywords in this domain, we retrieve 15 matching articles. We then set the configuration parameter *depth* to 1. Following this, we collect for each article that matches a keyword the articles in the respective categories (see Figure 3.10). Finally, we collected a total of **686 articles**, which are considered as our domain-specific corpus.

We apply WikiDoMiner on SRS4 (from the transportation domain) in a similar manner. The two resulting corpora are depicted in Figure 3.11 as word clouds. We show for each domain the main terms that frequently occur in the corpus. We see that the keywords *rail*, *track*, *train*, *railway*, and *railroad* characterize the railway corpus, while the transportation corpus is characterized by the keywords *traffic*, *road*, *street*, and *lane*. We note that the railway domain can be regarded as a sub-domain of the transportation domain. This observation is highlighted through the frequent terms that the two corpora have in common in Figure 3.11, such as *signal*, *system*, *vehicle*, and *driver*.

Domain Representativeness To examine how representative the resulting domain-specific corpora are, we compute semantic relatedness as follows. We first transform each article in the corpus into a vector representation using word2vec. We do the same for the test SRS. Then, we compute the cosine similarity between the vector representing the (test) SRS and the vector representing each article. In the following, we report the minimum, average, and maximum cosine similarity scores for each domain:

- Railway domain (cosine similarity between the railway corpus and test SRSs): min=0.27, **average=0.94**, and max=0.98
- Transportation domain (cosine similarity between the transportation corpus and test SRSs): min=0.67, **average=0.95**, and max=0.99.

Our results show that the domain-specific corpora are, on average, highly similar to the test (unseen) SRSs not used for generating the corpora. In particular, the average semantic similarity is ≥ 0.94, indicating that many articles in the corpus are relevant to the test SRSs. The minimum score of 0.27 in the railway domain implies that there are articles in the corpus which are more document-specific, i.e., more relevant to the SRS that induced the corpus but having little in common with the test SRSs. Note that, despite some document-specific articles

being present in the generated corpus, the very high average semantic similarity (≥ 0.94) indicates that such articles are a small minority and thus do not have a significant negative impact on the in-domain usability of the generated corpus.

The gap seen between the minimum scores reported for the two domain-specific corpora can be explained by the following: As mentioned in Section 3.5.2, all SRSs from the transportation domain in our collection are on the topic of traffic and transportation information management. This leads to extracting many keywords related to information management. In contrast, the SRSs in our collection from the railway domain are tailored to more specific topics, namely train control and digital radio standard for railway. This in turn leads to extracting document-specific terms which are related to train control (i.e., the topic of the SRS used for corpus generation) but not so much to digital radio standard for railway (i.e., the topic of the test SRSs). To summarize, our experiments show that WikiDoMiner has successfully generated representative corpora for two distinct domains.

3.6 Validity Considerations

Internal Validity. Bias is a potential threat to the internal validity of our evaluation. To mitigate this threat, the authors had no involvement in the annotation activities. Instead, two third-party annotators, who had no knowledge of our technical approach, independently annotated the dataset. Further, we made a strict separation between the data used for defining patterns and tuning, and the data used for assessing effectiveness.

Construct Validity. An individual requirement can potentially have multiple instances of CA or PAA. To ensure that this possibility is properly reflected in our metrics, we defined accuracy, precision and recall at the level of segments rather than whole requirements.

External Validity. Our evaluation builds on 20 industrial requirements documents, covering seven different domains. The promising results obtained across these domains provide a measure of confidence about the generalizability of our approach. This confidence is further strengthened by the fact that our approach can adapt itself to new domains via the (automatic) generation of domain-specific corpora. Due to this characteristic, we are optimistic that our approach will be able to achieve comparable results in other domains. That said, future case studies would help further improve external validity.

3.7 Conclusion

In this chapter, we proposed an automated approach for improving the handling of coordination ambiguity (CA) and prepositional-phrase attachment ambiguity (PAA). The main novelty of our approach is in automatically extracting domain-specific corpora from Wikipedia and utilizing them to increase the accuracy of CA and PAA handling in requirements documents. We conducted a large-scale evaluation of our approach using more than 5000 industrial requirements from seven different application domains. Our results indicate that our approach can detect CA and PAA with an average precision of $\approx$80% and an average recall of $\approx$89%. The results further indicate that employing domain-specific corpora has a substantial positive impact on the accuracy of CA and PAA handling. Specifically, over our dataset, we observed a $\approx$33% improvement in accuracy when compared to baselines using generic corpora. While our work is motivated by improving the quality of systems and software requirements, our technical solution is also novel from an NLP standpoint. Our solution thus has the potential to be useful over other types of textual documents within and beyond software engineering.

In future work, we would like to integrate our ambiguity handling approach with automated techniques for extracting structured information from requirements specifications. The motivation for doing so is to increase the quality of information extraction by more accurately interpreting coordination and prepositional-phrase structures. Another direction we would like to explore in the future is to use deep learning to complement or as an alternative to our current approach.

Chapter 4

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

4.1 Motivation and Contributions

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* [133]. Anaphoric ambiguity occurs when there is more than one plausible antecedent [35, 9]. In linguistics, there are several types of anaphora [133]. In requirements engineering (RE), anaphora is typically scoped to pronominal anaphora, i.e., when the anaphor is a *pronoun* [21, 2]. This is because pronominal anaphora has been clearly established as a genuine source of ambiguity in requirements [31]. *Anaphoric ambiguity detection* in RE is thus the task of identifying ambiguous occurrences of pronouns [134]. The closely related task of *anaphora resolution (interpretation)* is concerned with finding the most likely antecedent for a given pronoun [35].

To illustrate, consider the example in Figure 4.1. 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.

Anaphoric ambiguity is prevalent in NL requirements. Estimates from the RE literature suggest that nearly 20% of industrial requirements contain anaphora [1, 2]. Current RE research on anaphoric ambiguity [134, 21, 135, 136, 2], as we elaborate in Section 4.2.2, does not adequately explore two important facets. First, the

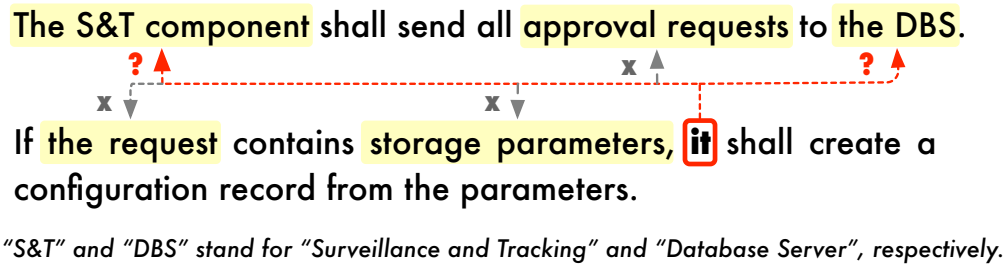


Figure 4.1: Example of Anaphoric Ambiguity.

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 [23] 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 software requirements specifications.

Our aim in this chapter 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 chapter 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 chapter 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 [137], 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 [138], 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 software requirements specifications from eight different application domains and containing a total

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

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

Structure. The remaining of this chapter is organized as follows. Section 4.2 discusses background and positions our work against the related literature in NLP and RE. Section 4.3 presents our alternative solutions for handling anaphoric ambiguity in requirements. Section 4.4 reports on our empirical evaluation. Section 4.6 addresses threats to validity. Section 4.7 concludes the chapter.

4.2 Background and Related Work

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

4.2.1 Background

Below, we discuss the enabling technologies used by our solutions marked ① to ⑥ in Figure 4.3. The precise design of these alternative solutions will be elaborated in Section 4.3.

Language Models (LMs). 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 [139]. SpanBERT is a variant of BERT that is optimized for the prediction of spans of text. In contrast to BERT, SpanBERT is pre-trained to predict masked text spans (rather than masked tokens). SpanBERT is trained only on one objective, that is, the span boundary objective (the start and end of the text span boundary). SpanBERT is better suited than BERT for text span prediction and selection 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 [140]. 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 [23] and SentenceBERT (SBERT) [57]. While BERT derives embeddings for individual tokens, SBERT is optimized for deriving semantically meaningful embeddings for an entire text sequence. Necessary background about LMs, BERT, SpanBERT, word embeddings, and SBERT is provided in Chapter 2.

Machine Learning (ML). 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 4.3. 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 Chapter 2 for more details on these algorithms.

Natural Language Processing (NLP) Pipeline. In our work, we apply an NLP pipeline composed of eight modules: (1) *tokenizer*, (2) *sentence splitter*, (3) *part-of-speech (POS) tagger*, (4) *lemmatizer*, (5) *constituency parser*, (6) *dependency parser*, (7) *coreference resolver*, and finally, (8) *semantic parser*. Necessary background about these NLP pipeline modules is provided in Chapter 2. Modules 1 to 6 are prerequisites for all our solutions (see the preprocessing step in Section 4.3.2). Our ML-based solutions additionally use modules 7 and 8 for extracting language features. Module 7 is the basis for solution ⑥.

4.2.2 Related Work

Ambiguity in natural language has been studied extensively [85, 35, 32]. In RE, different dimensions of ambiguity have been explored, including understanding the significance of ambiguity in requirements [82, 9, 32, 81, 83], analyzing the linguistic causes of ambiguity [31, 84, 19, 85], ambiguity prevention [14, 15, 16, 17, 18], and ambiguity detection and resolution [60, 86, 21, 75, 17, 87, 1, 141, 2, 88, 89, 142, 143, 10]. 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 [32]. Yang et al. [21, 134] 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 $\approx$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 [17, 1, 32].

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 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. [134, 21], enhanced with several new language features gleaned from the literature [144, 145, 146, 147, 148, 149]. 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 [35]. 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 [150, 149]. Despite numerous attempts at addressing anaphora resolution, the complex nature of the task has slowed progress for several anaphora types [151]. The anaphora resolution techniques in the NLP community are broadly classified into three categories: syntactic, semantic and neural-network-based [152]. The syntactic and semantic approaches focus on designing ML features based

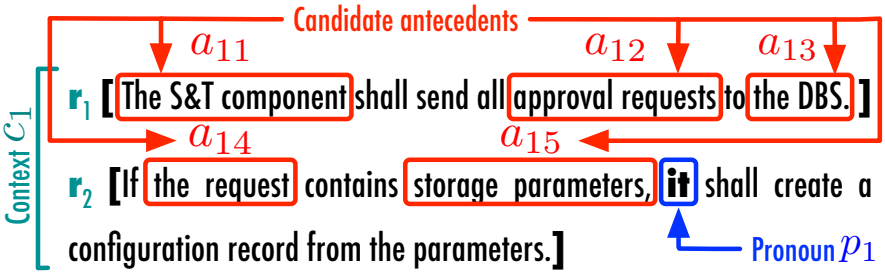


Figure 4.2: Illustration of our Notation.

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 [153, 154].

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 [132], NLP tools usually do not work well if applied as-is to requirements documents [143, 10]. 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.

4.3 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 4.4.

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 [21], 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 4.1. 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

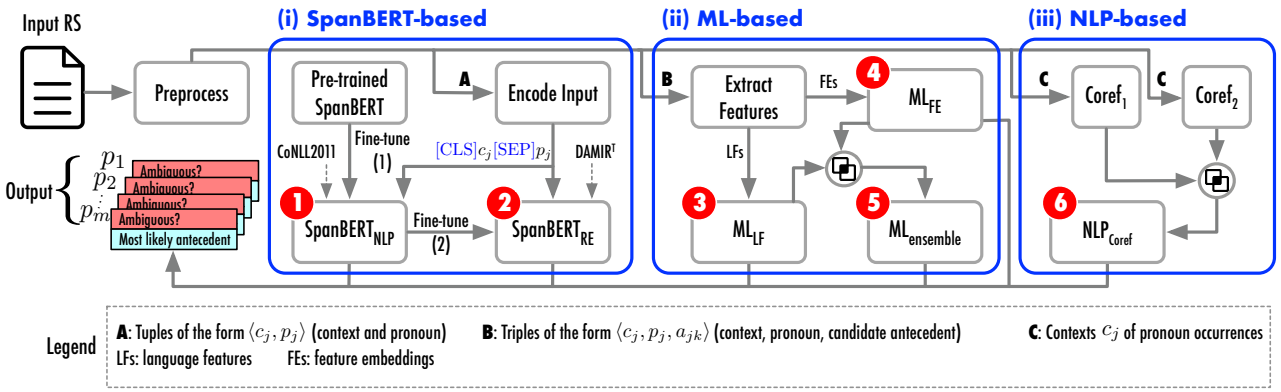


Figure 4.3: Overview of Solution Alternatives (marked ① to ⑥).

a_{15} = *"storage parameters"*. For easier referral later in the chapter, we visually show in Figure 4.2 how our notation is applied to the example of Figure 4.1.

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.

4.3.2 Preprocessing

The preprocessing step generates the input for the alternative ambiguity handling solutions that we consider in this chapter. We first apply the NLP pipeline, discussed in Section 4.2.1, on a given software requirements specification (SRS) to parse its textual content. We create the list of all pronouns (i.e., $\mathcal{P}$) occurring in SRS; this is done by selecting the words that the POS tagger marks as *PRP* (personal pronoun) or *PRP*$ (possessive pronoun) [155]. 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 4.1, 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 [21, 101].

4.3.3 Alternative Solutions

We consider six alternative solutions for handling anaphoric ambiguity. These are shown in Figure 4.3. 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 4.1 outlines for each solution the inputs, the intermediate outputs and the rules for processing the intermediate outputs. The rules produce the final results for anaphora resolution and ambiguity detection. We elaborate our alternative solutions next.

Table 4.1: 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 $\ell_{jx} =$ *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 $\ell_{jx} =$ *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.*

(i) Solutions based on SpanBERT. We employ the recent language model SpanBERT [139], introduced in Section 4.2.1, 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 [156, 157, 158] – a large dataset of generic text with about 7,000 pronoun occurrences. This fine-tuning step – fine-tune (1) in Figure 4.3 – 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 4.3 – 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 4.4.3.

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 $[\textit{CLS}]c_j[\textit{SEP}]p_j$. 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 $[\textit{CLS}]c_j[\textit{SEP}]p_j\#d$. 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 4.1. There is a threshold θ_α for controlling the resolution results. We recommend $\theta_\alpha = 0.9$ based on our

tuning, discussed in Section 4.4.5. For the example in Figure 4.2, 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 [21, 144, 145, 146, 147, 148, 149]. The description of the LFs is provided online [159]. ML_{FE} is trained on *feature embeddings (FEs)* which are contextual representations of the input, as explained in Section 4.2.1. $\text{ML}_{ensemble}$ 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 [23] and SBERT [57]. The FEs capture the semantic and syntactic regularities of a text sequence [160]. There are other pre-trained models, e.g., GloVe [161] and word2vec [55], 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 [162]. In Section 4.4.5, 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 4.4.5, 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. [21] and provided in Table 4.1. 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 4.4.5. To illustrate the ML-based solutions, recall the example of Figure 4.2. 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 4.4.5, 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 4.3, 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 4.1, combines the results of $Coref_1$ and $Coref_2$ via consensus. We instantiate $Coref_1$ and $Coref_2$ using two popular coreference resolvers [163]: the resolver in the CoreNLP toolkit [164, 165] and the one in the SpaCy library [128]. For the example of Figure 4.2, NLP_{coref} resolves p_1 as referring to a_{14} ("the request"), thus deeming p_1 as *unambiguous*.

4.4 Empirical Evaluation

In this section, we tune and assess the alternative solutions presented in Section 4.3.

4.4.1 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 4.3, 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 4.3, 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 [166, 167], one would typically want to use the results of automated anaphora resolution as-is and without additional manual processing.

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 4.3.

4.4.2 Implementation and Availability

We use Python 3.8 [126] for implementing the preprocessing step (Section 4.3.2) as well as for the high-level scripting of the alternative solutions shown in Figure 4.3. The NLP pipeline and language-feature extraction are implemented using SpaCy 3.0.5 [128], NLTK 3.5 [127], Stanza 1.2 [168], and CoreNLP 4.2.2 [169]. The SpanBERT-based solutions use the Transformers 4.6.1 library [170] provided by Hugging Face (`https://huggingface.co/`) and operated in PyTorch [171]. For the ML-based solutions, we use Scikit-learn 0.24.1 [130]. 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-*

Table 4.2: 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	-

v2 model [172], 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 [128] and CoreNLP 4.2.2 [164, 165]. The different solutions proposed in this chapter are implemented using Jupyter Notebooks [137].

4.4.3 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* [156, 157, 158], the NLP dataset on coreference resolution released in the 2011 edition of the Computational Natural Language Learning conference (CoNLL2011); and *ReqEval* [173], 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 4.2 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 software requirements specifications (SRSs) 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 SRSs, 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) [174], we obtain an inter-rater agreement of $\kappa = 0.54$, which indicates moderate agreement [117] between the annotators. We note that for datasets related to ambiguity analysis, this level of agreement is to be expected [10], 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 4.4, 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 4.3.2). 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.

Table 4.3: Accuracy of Different Configurations of Solutions ③ and ④ for Anaphoric Ambiguity Detection.

Models		③	④			
		LF	FE_1	FE_2	FE_3	FE_4
DT	P	50.9	51	49.8	51.9	56.9
	R	94	86.3	89	89	87.3
	F_2	80.3	75.7	76.7	77.7	77.5
FNN	P	50.2	51.6	51.1	49.9	55.5
	R	96.2	99.2	98	94	96.9
	F_2	81.2	83.6	82.7	79.8	83.6
kNN	P	50.3	50.2	49.8	50.2	55.7
	R	91	98.2	96.8	97.7	98.3
	F_2	78.3	82.4	81.4	82.1	84.7
LR	P	49.5	50.3	50.4	51.2	52.5
	R	89	95.3	95.9	97.2	90.4
	F_2	76.6	80.6	81.2	82.3	78.3
NB	P	50.4	50.8	50.1	50.6	55
	R	99.7	98.3	96.9	98.7	95.3
	F_2	83.3	82.8	81.5	82.9	82.8
RF	P	49.9	50.3	51.1	50.6	57.7
	R	94.3	94.5	96.7	95.5	100
	F_2	80	80.3	82	80.9	85.9
SVM	P	50	50	50.4	50.4	56
	R	94.4	97.2	97.7	97.7	92.6
	F_2	80.1	81.7	82.2	82.2	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.

4.4.4 Evaluation Metrics

Anaphoric ambiguity detection. We evaluate ambiguity detection using *precision (P)*, *recall (R)* and F$_\beta$*-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 [21, 175], 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.

4.4.5 Solutions Tuning

In this section, we describe the tuning of our solutions. The resulting tuned solutions are used in Section 4.4.6 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 [176, 23, 139]. To generate SpanBERT$_{NLP}$ (solution ① in Figure 4.3), 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 4.3) 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 4.3.3). 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 4.3) 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) [21, 177, 178]. Following best practice [23, 57], 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 [179].

We evaluate all configurations using 10-fold cross-validation [110]. 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 4.3 and 4.4 list the various configurations and the results obtained for each. We note that, in Table 4.4, we apply the *full matching* 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*

Table 4.4: 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

Table 4.5: Accuracy Results for Different Anaphoric Ambiguity Handling Solutions.

	P, R, and F_2 of Ambiguity Detection						SR (%) of Anaphora Resolution			
	DAMIR^E			ReqEval			DAMIR^E		ReqEval	
	P (%)	R (%)	F_2 (%)	P (%)	R (%)	F_2 (%)	Full	Partial	Full	Partial
SpanBERT$_N LP$	40.0	81.8	67.6	60.2	75.8	72.1	73.5	88.2	**97.8**	97.8
SpanBERT$_R E$	36.9	77.2	63.3	57.6	96.8	85.2	68.9	**96.1**	**97.8**	**100**
ML$_L F$	57.6	**100**	87.2	61.8	98.9	88.3	81.2	-	57.4	-
ML$_F E$	57.5	**100**	87.1	**62.2**	98.2	88.0	51.0	-	82.9	-
ML$_{ensemble}$	**58.2**	**100**	**87.5**	61.5	**100**	**88.9**	**82.3**	-	57.4	-
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 (SR) are highlighted in **bold**.

algorithm trained over FE_1 for anaphora resolution. Following the above decisions, we apply grid search [180] 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 4.3.3). 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 $\theta_\delta = 0.1$.

4.4.6 Answers to the RQs

RQ1. *Which solution alternative is the most accurate for detecting anaphoric ambiguity in requirements?* Table 4.5 (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 4.5, 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 $\text{ML}_{ensemble}$ 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 $\text{ML}_{ensemble}$ in terms of precision. Across *ReqEval* and *DAMIR*^{E}, $\text{ML}_{ensemble}$ 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 *$\text{ML}_{ensemble}$ (solution ⑤ in Figure 4.3) 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 4.5 (right side) shows the resolution success rate (defined in Section 4.4.4) for *DAMIR*^{E} and *ReqEval*. Our evaluation covers 96 unambiguous pronouns in *DAMIR*^{E} and 62 in *ReqEval*. We apply both the full and partial matching modes (see Section 4.4.4). 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 4.5 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. $\text{ML}_{ensemble}$ 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 4.3) 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 4.4.5. Tuning is a *one-off activity* and not pertinent to end-users. We used Google Colaboratory [181] 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 4.4.5 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* $\textit{SpanBERT}_{NLP}$ *or* $\textit{SpanBERT}_{RE}$*; 8s for detection and 8s for resolution using* $\textit{ML}_{LF}$*; 7.5s for detection and 6s for resolution using* $\textit{ML}_{FE}$*; 14.5s for detection and 13s for resolution using* $\textit{ML}_{ensemble}$*; and 7s using* $\textit{NLP}_{Coref}$.

The practical implication of these execution times is as follows: Based on the literature [2], 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 $\text{ML}_{ensemble}$ 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.

4.4.7 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, $\text{ML}_{ensemble}$ 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

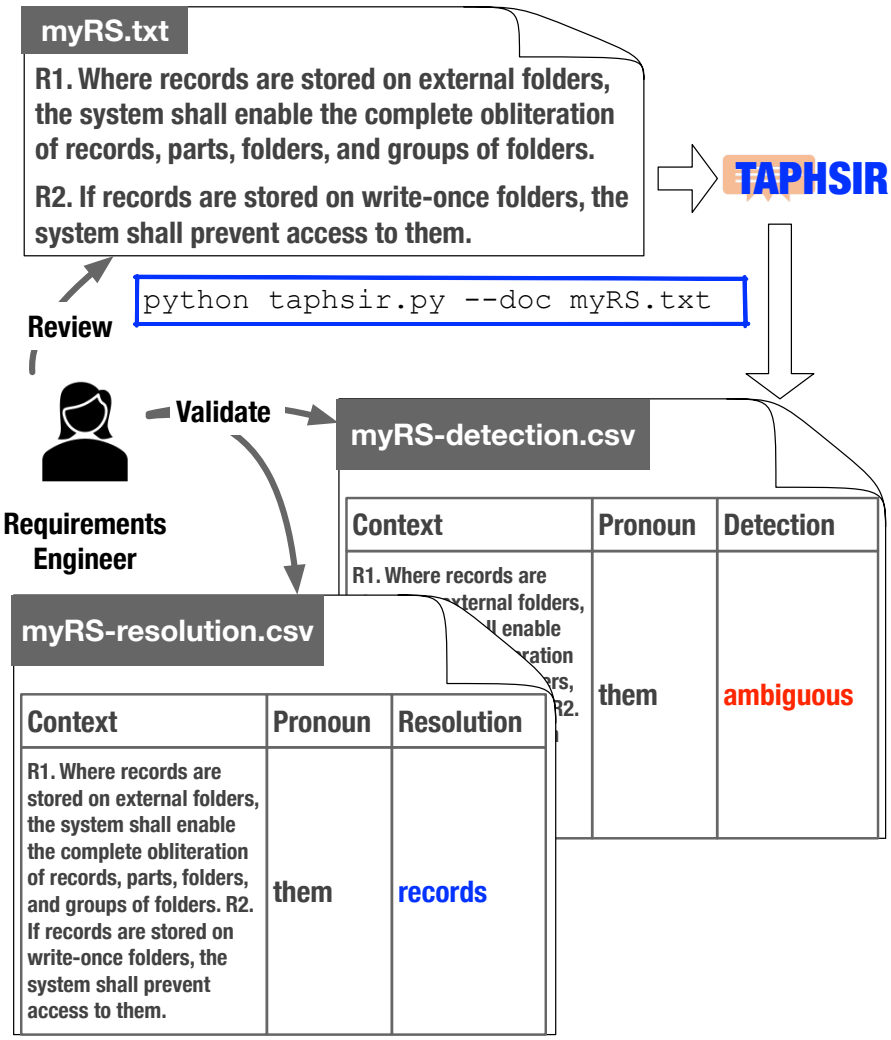


Figure 4.4: Application Example of TAPHSIR.

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.

4.5 Tool Support

TAPHSIR, standing for **T**owards **A**na**ph**oric Ambiguity Detection and Re**s**olution **i**n **R**equirements, is the implementation of the best performing parameters of our approach. In Arabic, TAPHSIR means "interpretation". TAPHSIR focuses on pronominal anaphoric ambiguity, an ambiguity type that has been explored only to a limited extent in requirements engineering (RE) [21, 2]. There are no existing tools in RE to handle anaphoric ambiguity, although this type of ambiguity is prevalent in NL requirements: it is estimated that up to 20% of industrial requirements may suffer from anaphoric ambiguity [1, 2].

TAPHSIR aims to reduce the time and effort that requirements engineers spend inspecting the use of pronouns in an SRS. To illustrate, consider the example in Figure 4.4. The figure shows a requirements engineer

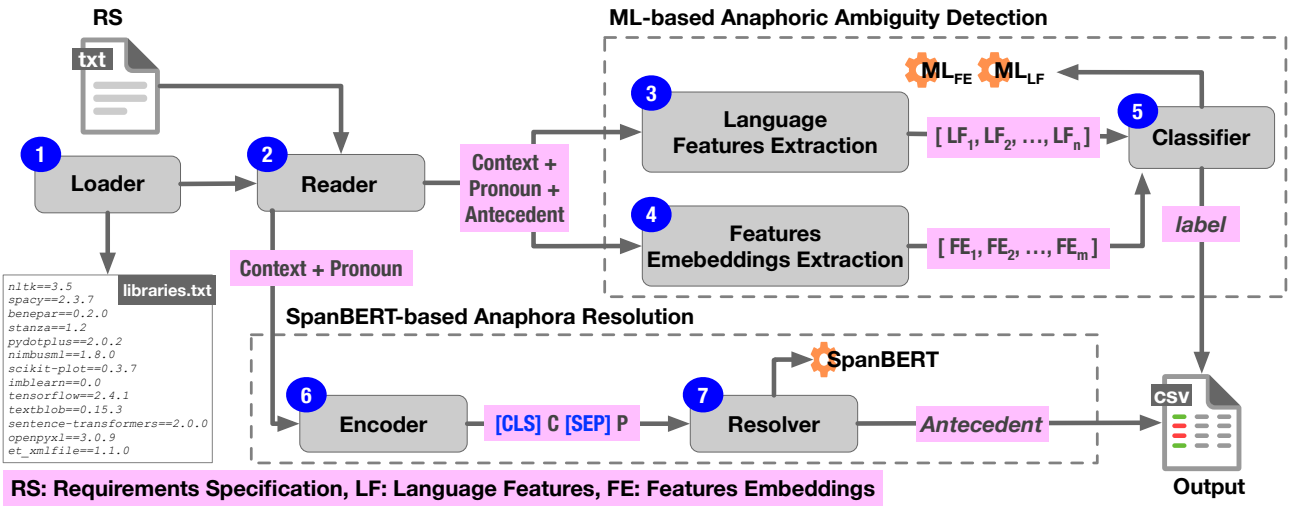


Figure 4.5: Overview of TAPHSIR Architecture.

reviewing the requirements in the file "mySRS.txt" and using TAPHSIR for automated analysis of pronominal anaphora in that SRS. The pronoun "them" in R2 contains anaphoric ambiguity since it is not clear whether the pronoun refers to the *write-once folders* (in R2), *records* only (in R1), or *records, parts, folders and groups of folders* altogether (in R1). Deciding about the exact interpretation has an impact on properly implementing the requirement. TAPHSIR defines a *context* for each pronoun occurrence. This context is composed of the requirement in which the pronoun occurs and the preceding requirement. Within this context, the tool identifies all noun phrases (NPs) preceding the pronoun as candidate antecedents [35]. In our example, TAPHSIR will consider, in addition to those mentioned above, the following candidate antecedents: *access*, *obliteration*, *system*. TAPHSIR then goes through different steps as we explain in the next section, and produces an output file ("mySRS.csv" in Figure 4.4). This output lists all pronoun occurrences in the input SRS, and provides both the detection decision as well as the resolution result. We note that TAPHSIR can recommend a resolution also for those pronouns that are marked as ambiguous, since it applies two separate solutions for detection and resolution.

TAPHSIR is a usable prototype tool for anaphoric ambiguity handling. The tool realizes a technical solution that resulted from an empirical examination of several alternative solutions [11]. Figure 4.5 shows an overview of TAPHSIR architecture. The tool is implemented in Python 3.8 [126]. Below, we discuss an end-to-end application of the tool going through steps 1 – 7 of Figure 4.5.

4.5.1 Preparation

Prior to using the tool, the user needs to perform some preliminary setup. To do so, one can type in the following on the command line:

```
python pip install -r libraries.txt
python -m spacy download en_core_web_sm
```

The first command installs the required libraries, and the second one downloads *en_core_web_sm* which is needed for operationalizing the natural language processing pipeline in SpaCy. To be able to apply the tool, the user further needs to ensure that the input file is in the right format. TAPHSIR expects as input a text file (with the extension **.txt*) containing a set of requirements (or sentences).

4.5.2 Reader

This step parses the text of the input requirements specification, preprocesses it using an NLP pipeline, and identifies the requirements that should be subject to anaphoric ambiguity analysis. The NLP pipeline consists of the following seven modules illustrated in Figure 4.5: (i) tokenizer splits the input text into tokens, (ii) sentence splitter demarcates the sentences in the text, (iii) part-of-speech tagger (POS) assigns a POS tag for each token, (iv) lemmatizer identifies the canonical form of a token, (v) constituency parser identifies the structural units of sentences, (vi) dependency parser defines the grammatical dependencies between the tokens in sentences, and (vii) semantic parser extracts information about words' meanings.

The output of this step is a set of *triples*, each of which includes a (i) a *pronoun* occurrence, (ii) *context* defined as the requirement in which the pronoun occurs and a preceding requirement (recall from Section 4.1, and (iii) a likely *antecedent* to that pronoun occurrence. The number of triples depends on the number of likely antecedents. In Figure 4.4, there are three possible antecedents as discussed in Section 4.1, namely "records, parts, folders and groups of folders", "records", and "write-once folders". Following this, this steps generates three triples associated with the pronoun occurrence "them", where each triple includes one possible antecedent. The triples will further have the same context, which combines R1 and R2.

ML-based Anaphoric Ambiguity Detection

Our earlier work [11] indicates that, for the task of anaphoric ambiguity detection, (feature-based) ML leads to better accuracy than language modeling and off-the-shelf NLP methods. For anaphoric ambiguity detection, we employ an ensemble ML classifier that combines the results of a classifier trained over language features (ML_{LF}) and another trained over feature embeddings (ML_{FE}). For training and applying ML classifiers, we use Scikit-learn 0.24.1 [130]. This component takes as input a set of triples associated with one the pronoun occurrence from the previous step, and derives as output a final label for that pronoun (ambiguous or unambiguous).

4.5.3 Language Features Extraction

This step extracts the different sets of learning features. In our work, we collected a set of 45 language features (LFs) from the NLP and RE literature. These features capture the characteristics of the relationship between the pronoun and its likely antecedent, e.g., both agree in gender or number. For extracting LFs, we use SpaCy 3.0.5 [128], NLTK 3.5 [127], Stanza 1.2 [168], and CoreNLP 4.2.2 [169]. The result of this step is a vector representing each input triple, where each entry in this vector is the result of computing an LF. For the example in Figure 4.4, we will generate three vectors representing the LFs of the pronoun "them" and each of its likely antecedents.

4.5.4 Extraction of Features Embeddings

This step extracts the feature embeddings (FEs) for each input triple. FEs are mathematical vectors that encapsulate the semantic and syntactic regularities of the sentence [131]. In our work, we extract 768 dimensional FEs from the BERT language model [23]. For that, we use the Transformers library, particularly the *bert-base-cased* model. Similar to the previous step, the output of this step is a vector representing each input triple. In a similar manner, this step results in three vectors for the example in Figure 4.4.

4.5.5 Classification

In this step, we pass the vector representation of each input triple to two pre-trained classifiers, namely ML_{LF} that is trained over LFs, and ML_{FE} trained over FEs. For each triple, the two classifiers independently predict a label as follows: *correct* (conversely, *incorrect*) indicating that the antecedent refers (conversely, does not refer) to the pronoun, or *inconclusive* when the anaphoric relation cannot be inferred. We then apply a set of rules on the predicted labels for the triples associated with one pronoun occurrence to conclude whether the pronoun is deemed ambiguous or unambiguous by each of the two classifiers. The rules, presented in the RE literature [21], consider the prediction probabilities produced for each possible antecedent.

Finally, we combine in an ensemble manner the results of the two classifiers ML_{LF} and ML_{FE} to derive the final label for the pronoun (i.e., ambiguous or unambiguous). If the two classifiers agree on the label (e.g., both conclude that the pronoun is ambiguous), then this label will be the final one for that pronoun. Otherwise, the label with the highest prediction probability will be selected. This ensemble learning method yields a more accurate prediction.

SpanBERT-based Anaphora Resolution

Based on the empirical findings in our earlier work [11], we know that for the task of anaphora resolution, the SpanBERT language model [139] outperforms alternatives. Consequently, the resolution component in TAPHSIR uses a SpanBERT model that is fine-tuned on a curated dataset from requirements. The dataset will be discussed in the next section. We implement SpanBERT using the Transformers 4.6.1 library [170] provided by Hugging Face (`https://huggingface.co/`) and operated in PyTorch [171]. This model takes as input, from the triples generated in the first step, only the pronoun and the context in which it occurs (i.e., disregards the likely antecedents). As SpanBERT is originally trained to extract text spans, SpanBERT in our work predicts as output the likely antecedent for the pronoun from its context.

4.5.6 Encoder

To be able to use SpanBERT model, the input pair of context and pronoun has to be encoded into the same format as the training data that BERT has been trained on. To do so, the input tuple is passed on to BERT's tokenizer which adds two special tokens: *[CLS]* to represent the classification output and *[SEP]* to separate the context from the pronoun occurrence. The token *[SEP]* informs BERT about which pronoun occurrence to analyze in the given context.

4.5.7 Resolver

In this step, we pass on the encoded input to the fine-tuned SpanBERT model and have the model predict the text span which likely represents the antecedent of the pronoun. SpanBERT can predict multiple such text spans with different probabilities indicating the likelihood of being the right antecedent. If an antecedent is predicted with a high probability (greater than 0.9), then we consider this as the resolution result for the pronoun.

Output

Given an input SRS, the output of our tool is a csv file listing all pronoun occurrences in the input, and for each occurrence, providing the predicted label (ambiguous or unambiguous) and the most probable antecedent.

4.5.8 Evaluation

In this section, we evaluate how accurately TAPHSIR can detect unacknowledged cases of anaphoric ambiguity and bring them to the attention of the requirements engineer.

Dataset Description

In this section, we use the curated dataset *DAMIR* (standing for Dataset for Anaphoric Ambiguity In Requirements) [11]. We curated this dataset with the help of two third-party annotators who underwent half-day training on ambiguity in requirements. We collected 22 industrial requirements specifications covering eight domains including satellite communications, medicine, aerospace, security, digitization, automotive, railway, and defence.

We preprocessed this collection and prepared the list of triples (a context, a pronoun occurrence and a possible antecedent) as explained above. The possible antecedents for a pronoun include all of the noun phrases preceding that pronoun [182]. The annotators then examined each pronoun occurrence and its possible antecedent considering the context in which they occur, and assigned a label *correct*, *incorrect*, or *inconclusive* with the same indications as explained above. We then post-processed the annotations and grouped them per pronoun occurrence as follows. We mark a pronoun as ambiguous in two cases: (i) if at least one annotator acknowledges the ambiguity of this pronoun by labeling one or more associated triples as *inconclusive*; or (ii) if the same triple associated with this pronoun receives different labels from the two annotators (e.g., *correct* versus *incorrect*). The former case implies acknowledged ambiguity, and the latter implies unacknowledged ambiguity.

As a result, *DAMIR* dataset contains a total of 737 pronoun occurrences that are analyzed for anaphoric ambiguity. About 46% of these pronouns (342/737) are deemed ambiguous by the annotators. Out of the ambiguous pronouns, we identified ≈87% with unacknowledged ambiguity, i.e., the annotators assumed that the pronoun is unambiguous yet had two different interpretations for that pronoun.

Results and Analysis

To assess how TAPHSIR performs in detecting unacknowledged ambiguity, we run TAPHSIR (depicted in Figure 4.5) on *DAMIR* dataset. TAPHSIR applies the an ensemble ML classifier for detecting ambiguity and SpanBERT for resolving anaphora as discussed above. On *DAMIR* dataset, TAPHSIR detects ambiguous cases with a perfect recall of 100% with a precision of ≈60%, while recommends automated resolution with an accuracy of ≈96% [11]. The perfect recall implies that TAPHSIR detects all unacknowledged ambiguous cases that were not explicitly marked by the human annotators as ambiguous. The precision value indicates that the requirements engineer will invest some manual effort filtering out false positives, i.e., falsely detected ambiguous requirements. In the context of ambiguity in RE, recall is often favored over precision [183]. Achieving 100% recall ensures that all requirements suffering from all potentially ambiguous requirements will be brought to the attention of the engineers and further discussed at an early stage.

In a practical scenario where requirements engineers review requirements under time pressure, only the requirements that are found problematic by at least one engineer would be thoroughly discussed. The engineers might not discuss those requirements which they could confidently interpret unaware of having multiple inconsistent interpretations. In conclusion, we believe that TAPHSIR has a potential in practice since it perfectly detects also those requirements with unacknowledged ambiguity which would go otherwise unnoticed during manual inspection sessions. That said, a user study is required to assess the practical usefulness of the tool.

4.6 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.

Construct Validity. Due to their different enabling technologies, the solutions examined in this chapter require different inputs. Notably, the ML-based solutions take as input triples comprised of a pronoun, a likely antecedent and a context. In contrast, the SpanBERT-based solutions take as input tuples made up of a pronoun and a context. We believe that exposing the ML-based solutions to candidate antecedents during training does not put SpanBERT-based solutions at disadvantage: The SpanBERT-based solutions are exposed to candidate antecedents during fine-tuning. Since SpanBERT is pre-trained to identify text spans, we fine tune the model to predict the text span that represents likely antecedent of a pronoun. A requirements sentence might have multiple occurrences of the same pronoun. We ensure that the detection and resolution of an ambiguous pronoun is properly reflected in our evaluation by treating each occurrence as a distinct pronoun.

4.7 Conclusion

In this chapter, 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.

Chapter 5

AI-based Question Answering Assistant for Analyzing Natural-language Requirements

5.1 Motivation and Contributions

At early stages of software development, natural language (NL) requirements are reviewed to ensure their quality and reduce any potential misunderstandings that might have an impact on the software development process. Among quality issues that requirements engineers have to deal with are different types of ambiguity [9, 10], incompleteness [12], inconsistency and change analysis [184]. Depending on the complexity of the system-to-be, an SRS can contain hundreds (and even thousands) of requirements. Therefore, manual inspection where requirements engineers attempt to discuss and interpret *problematic* requirements is time-consuming and error-prone. We refer to the requirement that suffers from at least one quality issue as a *problematic* requirement.

In this chapter, we propose *ReQAssis* standing for Requirements Quality Assurance Assistance through Question Answering. *ReQAssis* is an AI-enabled automation that uses question-answering (QA) as a means to support requirements engineers in inspecting requirements. To better interpret certain requirements, the engineer might need to instantly browse through the different parts of an SRS in order to find relevant information. For example, the requirement "the system shall compute the wet mass of the spacecraft at a regular basis." misses information concerning the computation procedure of the wet mass and the exact frequency of doing this computation. Such information might be mentioned elsewhere in the SRS. In this case, it is advantageous if the analyst can pose questions like "how shall the wet mass be computed?" and "how often shall the wet mass be computed?", and get a prompt answer. This is a typical use of a QA assistant. Another potential use that benefits from QA is to cover the gap in domain-specific knowledge. Posing questions about the definitions of domain-specific terms to an automated assistant helps the requirements engineers build a glossary table for the SRS with minimal effort. Alternatively, posing a clarification question about unclear requirement helps the engineer acquire necessary domain-knowledge for a better understanding of the requirement.

To cover the potential uses above, we differentiate between a *document-based* question whose answer can be found in the same SRS, and a *domain-based* question whose answer is not explicitly specified in the SRS and

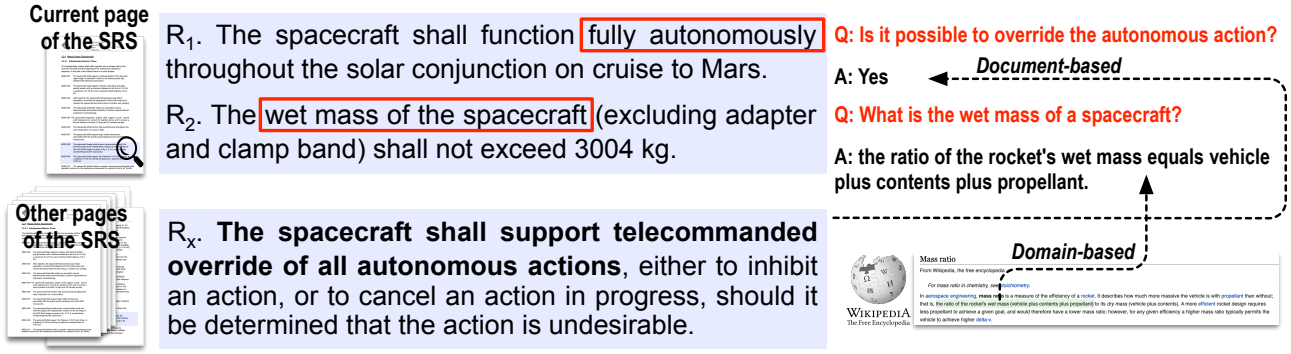


Figure 5.1: Excerpt from *Software Requirements Specifications*.

need to be looked up in an external knowledge resource for that domain. To illustrate the multiple scenarios for using the QA assistant, consider the example in Fig. 5.1. Assume that a requirements analyst is reviewing the current page of a given SRS. On that page, R1 describes the autonomous functionality of the spacecraft, while R2 describes the wet mass of the spacecraft. To properly implement R1, it is important to know whether a counter-requirement about an override functionality is also specified. This information is mentioned in Rx, which appears first in a later section. To properly interpret R2, one needs to understand what a wet mass of a spacecraft is. Since this information is not explicitly given in the SRS, the answer has to be mined from an external resource, e.g., a domain-specific corpus generated by crawling Wikipedia.

In RE, QA has been investigated mostly in the context traceability [24, 25, 26]. Traceability refers to the ability to follow the life of a requirement throughout the software development process [8]. A set of questions are often used to trace a requirement across multiple artifacts, e.g., system specifications, user stories, and class diagrams. Such trace questions are often generated to find related artifacts to a given requirement. Traceability can also be within an SRS, e.g., the engineer asks about requirements impacted by certain change in the SRS. Compared to existing work in RE, we exclusively focus on questions posed during inspection sessions of SRSs. We concern ourselves with clarification questions that help improve the interpretation of requirements and quality assurance of NL requirements.

In NLP, QA is a well-established topic. In our work, we adapt the definition of an open-domain QA, which is the task of finding the answer to a given question in a collection of documents [185]. Our approach (*ReQAssis*) combines two components: a *Retriever* which retrieves a document and/or a text passage in a document using information retrieval (IR) methods [186], and a *Reader* which extracts the answer to the question from the retrieved text passage. The latter is a standalone task in NLP known as machine reading comprehension (MRC) [187]. There is a wide spectrum of existing work with regard to QA as we elaborate in Section 5.5. NLP methods cannot be directly applied in our work due to the following limitations. First, there is no available dataset that is specific to RE such that the question-answer pairs describe the content of an SRS. Second, existing methods do not foster domain-based questions whose answers are found in a domain-specific corpus.

Contributions. We take steps toward addressing the above limitations. Concretely, this work has the following contributions:

(1) We devise *ReQAssis*, an AI-based automated quality assurance assistance by means of QA. *ReQAssis* takes as input an SRS and a question posed in NL. *ReQAssis* then employs two models: a *Retriever* and a *Reader* that jointly provide the requirements engineer with the output consisting of a list of relevant text passages and a potential answer highlighted in each passage for a given question.

(2) We generate a dataset for QA that is specific to NL requirements in a semi-automatic manner. We refer

to this dataset as *REQuestA* which stands for Requirements Engineering Question-Answering dataset. *REQuestA* contains a total of 387 question-answer pairs, of which are 173 automatically generated and the rest are proposed by human annotators. We created *REQuestA* with the help of two third-party annotators who validated the auto-generated questions and answers, and also proposed additional question-answer pairs. The two annotators have been involved in several other annotation tasks for RE, and gained sufficient experience from working on the same SRSs. We discuss the annotation process in more detail in Section 5.4.3. To develop *ReQAssis* and generate question-answer pairs automatically for *REQuestA*, we utilize in our work large-scale language models that are reported to perform well for QA [188].

(3) We empirically evaluate *ReQAssis* on *REQuestA* dataset. *REQuestA* is created from a collection of six SRSs covering three different domains, namely aerospace, defence and security. We experiment with different possible configurations for *ReQAssis* and report in Section 5.4 on the best performing configuration for each component. Our results indicate that *ReQAssis* (i) distinguishes between document- and domain-based question types with an accuracy of 88.5%, (ii) retrieves from domain-specific corpora of various sizes the document that contains the right answer to a given domain-based question with a perfect accuracy of 100%, (iii) retrieves the text passage that contains the right answer to the input question among the top-3 relevant text passages from the retrieved document in (ii) or the input SRS depending on the question type with a average accuracy of 90.6%, and finally (iv) extracts from the right passage the likely answer to a given question with a semantic accuracy of 84%. In Section 5.4, we present more details on the evaluation metrics and the results of our evaluation.

Structure. Section 5.2 presents the background of the language models that we use in our work. Section 5.3 describes *ReQAssis*. Section 5.4 reports on our empirical evaluation. Section 5.6 discusses threats to validity. Section 5.5 places our work against the NLP and RE literature. Section 5.7 concludes the chapter.

5.2 Background

In this section, we present a brief introduction to language models, followed by an explanation of different *Retriever* and *Reader* models we apply in our work.

Language Models (LMs) for QA. LMs are statistical models that assign to a sequence of words a probability value estimated from a large training corpus, indicating the likelihood of these words to co-occur in the sequence. For example, an LM would assign a higher probability to the phrase "briefed reporters on" compared to the phrase "briefed to reporters" [131]. Such probabilistic models are highly dependent on the training data. For solving different downstream NLP tasks, different probabilistic LMs are built using *in-domain* data, i.e., task-relevant corpora. For example, to build an LM for a question-answering system, the training corpus should contain questions, answers, and context information.

In many real-world applications, it is expensive to collect enough training data and rebuild statistical LMs for each and every NLP task. To address this issue, *transfer learning* can be used [189]. Specifically, LMs are *pre-trained* on a large amount of generic text. This pre-training process results in a model with general knowledge about the words and their syntactic and semantic relations. Then, the pre-trained LMs can be fine-tuned, even using small datasets, to solve specific NLP tasks. BERT [23] (Bidirectional Encoder Representations from Transformers) is a notable example of such pre-trained models, which have been widely applied for solving many downstream NLP tasks.

BERT is pre-trained on the BooksCorpus and English Wikipedia, with two training objectives, namely masked language modeling (MLM) and next sentence prediction (NSP). More details about LMs and BERT can

be found in the Chapter 2.

DistilBERT (a distilled version of BERT) [190], is lite version of BERT that uses distillation (a small student model trained to reproduce the behaviours of a larger teacher model) to produce a liter model with close performance to BERT. DistilBERT is based on BERT-base architecture with some components optimized or removed, and a reduced number of layers. This optimization techniques results of a lite model that is 60% faster and has 40% fewer parameters than BERT while retaining 97% of capabilities.

MiniLM (Mini Language Model) [191] is a similar model to DistilBERT, as it aims to produce a light yet efficient LM using the distillation technique. MiniLM (the student model) focuses on learning the the self-attention modules of the teacher model (BERT-base). Moreover, MiniLM utilizes an improved version of the self-attention distributions of the last Transformer layer of the teacher model. This results in a model that is cheaper (consumes less resources to apply) and powerful at the same time (MiniLM outperformes BERT-base and DistilBERT on some NLP benchmarks).

ALBERT (A Lite BERT) [192] is a variant of BERT that alters some parameters of BERT to increase its efficiency. It results in lower memory consumption and better computing speed compared to BERT. ALBERT is based on the extended version of BERT (BERT-large) with fewer parameters. ALBERT is not only more efficient than BERT-large, but also achieves better performance on some NLP benchmarks.

RoBERTa (A Robustly Optimized BERT Pretraining Approach) [193], as the name shows, is another variant of BERT that aims to optimize its training parameters and choices to improve its performance. Compared to BERT, RoBERTa authors proposed to train the model longer and on more text with larger batches. They also propose to change some pretraining objectives by removing the NSP objective and changing the training masking pattern. RoBERTa model outperforms BERT on some NLP benchmarks.

ELECTRA (Efficiently Learning an Encoder that Classifies Token Replacements Accurately) [194], is yet another variant of BERT that aims to increase its efficiency by improving the pretraining objective. The authors of ELECTRA propose to replace the token masking objective by token replacement method that corrupts some tokens. The model objective becomes then to predict for each token whether it was corrupted or not. The new pretraining method results in a faster training time and outperforms larger models on some NLP benchmarks.

T5 model (**t**ext-**t**o-**t**ext **t**ransfer **t**ransformer) model [195] has been more recently introduced. T5 is pre-trained on the Colossal Clean Crawled Corpus (C4) which was also released with the model. C4 consists of hundreds of gigabytes of clean text that is crawled from the Web. Unlike previous LMs, T5 is a text-to-text model, which means that it addresses an NLP task after it is transformed into a text-to-text problem. For example, text summarization is a text-to-text problem, since it takes a text as input and returns another text as output. For text classification tasks, the T5 model predicts as output a single word that corresponds to the label (e.g., "positive" or "negative" in a sentiment analysis task). However, T5 can also predict words that are not included in the predefined labels of the classification task (e.g., "food") which is then counted as a wrong prediction. There are publicly available T5 models that are pre-trained on different tasks. In our work, we use T5 models that are pre-trained on QA datasets specifically to generate question-answer pairs and build our dataset, as we elaborate in Section 5.3.

Retriever Models. *Retriever* models (*Retrievers* for short) apply IR-based methods. IR is the field concerned with identifying relevant documents or text passages for a query from a collection of documents [196]. In our work, we experiment with the best *Retrievers* reported in the recent Benchmark for IR (BEIR) [188]. This includes traditional approaches like TF-IDF and BM25, advanced methods based on recent large-scale language models like DistilBERT, and re-ranking methods. We explain these methods next.

(1) Term Frequency - Invert Document Frequency (TF-IDF): Necessary background on TFIDF is provided in Chapter 2. TF-IDF is often used as a ranking factor for information retrieval and text mining. The main applications of TF-IDF include text-based recommendation systems and search engines. The TF-IDF score is the product of the values of TF (term frequency) and IDF (inverse document frequency).

Despite TF-IDF being less used in the IR community in view of the advancements of NLP technologies, we still apply TF-IDF as a baseline in our experiments.

(2) Okapi Best Matching (BM25): Similar to TF-IDF method, BM25 estimates the relevance of a particular document to the query terms (question in our case) [197]. Compared to TF-IDF, BM25 is more sensitive towards term frequencies and document length. BM25 is a method that computes and assigns a score to each document according to its relevance to a given question. BM25 is still widely applied in the IR community [197].

Specifically, the BM25 score of a document D containing terms $q_1, q_2, .., q_n$ that are part of a given query Q is calculated based on the IDF value for each term q_i, the frequency of the term q_i in document D, the total number of words in document D, the average length of the documents in the text collection, and two free parameters that are usually selected without advanced optimization [198].

(3) DistilBERT In recent years, the IR domain has witnessed a shift towards using large-scale language models. These methods are also called dense *Retriever* models. We experiment in our work with DistilBERT [190] (explained above). Dense retrieval methods capture semantic relevance between the document and the question by embedding them into a shared dense embedding space using deep learning. Though dense *Retrievers* improved the accuracy of the state-of-the-art, they are often computationally expensive than the traditional methods.

(4) Reranking methods: Reranking methods are recently introduced to utilize the output of two different IR-based *Retrievers*. The idea is to let the first *Retriever* rank the document collection according to relevance to a given question, then use a second *Retriever* to rerank the top-K relevant documents. In our work, we apply the most commonly paired methods, namely BM25 followed by MiniLM cross-encoder [199]. Cross-encoder models are designed to provide a score for each input pair of sentences. This make cross-encoders suitable for the IR task, as they can be fine-tuned to provide scores and rank query and context input pairs according to the relevance of the context to the input query.

Readers. To solve the MRC task in NLP, *Readers* take a question and a text passage as input and return an answer by selecting a text span within the input passage. Readers can be classified into *Extractive Readers* that extract an answer span from the retrieved passages and *Generative Readers* that generate answers in NL using text-to-text or sequence-to-sequence (Seq2Seq) models.

(1) Extractive Readers: Probabilistic *Readers* uses similarity- and heuristic-based approaches to extract the answers [200]. However, recent Transformer-based LMs proved to be successful and demoniated the state-of-the-art (SOTA) for this task. Namely, bi-encoders such as RoBERTa [193], BERT [23], DistilBERT [190], ALBERT [192], ELECTRA [194], and MiniLM [191]. These models (explained above) are the best performing LMs for this MRC task [201, 202, 203].

(2) Generative Readers: The goal of using generative readers is to generate naturally formulated answers, rather than extracting text spans, usually relying on text-to-text models such as BART [204] and T5 [195]. However, Generative Readers need to be further explored and improved, as their results often suffer from syntax errors, inconsistencies, or illogicality [205]. Experimenting with generative *Readers* is left for future work.

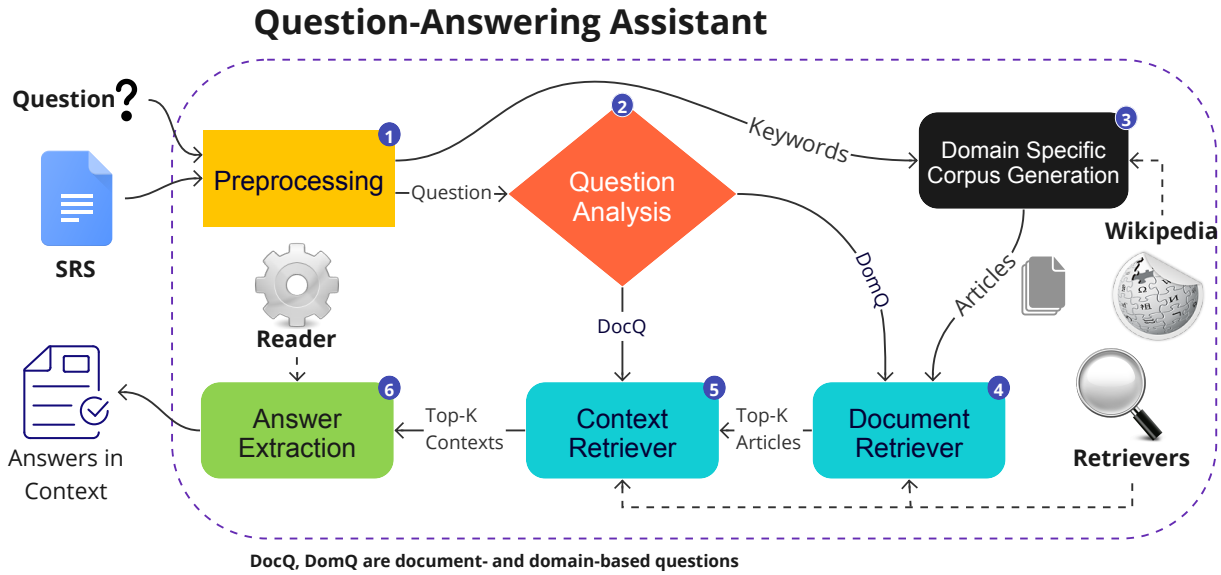


Figure 5.2: Overview of *ReQAssis*.

5.3 Approach

Our QA assistance approach (*ReQAssis*) is composed of six steps as depicted in Fig. 5.2. The input to the approach includes a software requirements specification (SRS) and a question (q) posed in NL by a human analyst. The output is a list of relevant text passages for q, in each is a possible answer highlighted. Thereafter, we will refer to the text passage as *context* c.

In step 1, we preprocess SRS using an NLP pipeline. In step 2, we analyze q and decide whether it is a document-based (*DocQ*) or domain-based (*DomQ*) question. Based on this decision, we either perform steps 3 and 4 or directly proceed to step 5. In step 3, we generate a domain-specific corpus by crawling Wikipedia based on relevant keywords extracted from the input RS. For readability purposes, we will refer to a document in the domain-based corpus as *article*, since in our work we build the corpus from Wikipedia. Subsequently, we retrieve in step 4 the most relevant document from this corpus that are relevant to q. In step 5, we search for contexts relevant to q within one document (RS for *DocQ* or the article retrieved in step 4 for *DomQ*). In the same step, we further rank the resulting set of relevant contexts and select the top-K. Finally, we extract in step 6 a potential answer to q from each relevant context resulted from the previous step. We note that steps 4 and 5 apply IR-based models whereas step 6 applies MRC-based models. In the following, we elaborate on each step.

5.3.1 Preprocessing

To preprocess RS, we apply a standard NLP pipeline consisting of two modules, namely *tokenization* and *sentence splitting* to break the text into tokens and sentences.

In this step, we further partition SRS into a set of contexts. We define a *context* as a text passage that contains consecutive sentences from SRS fitting a maximum tokens limit. The goal of this partitioning is to prepare a meaningful input for steps 5 and 6, as we explain later in this section.

5.3.2 Question Analysis

In this step, we analyze the input question q. In our work, we address domain-based questions for the purpose of glossary extraction (definition question) in addition to clarification questions. Such definition questions take the form of a predefined template: *"what is the definition of [concept]$_x$"*. Templated questions can be predicted using heuristics. However, to provide a more general solution, we train in this step a supervised ML-based classifier over tokenized questions to predict the question type, i.e., DocQ or DomQ. We experiment in Section 5.4 with several classification algorithms and different representations, namely TFIDF-vectors, and word2vec and SBERT embeddings. we perform steps 3 and 4 if q is predicted as DomQ, As shown in Fig. 5.2. Otherwise, we proceed to step 5.

5.3.3 Domain-specific Corpus Generation

This step generates a domain-specific corpus for SRS by crawling Wikipedia. To do so, we apply an existing automated method [10]. In summary, the method first extracts the top-N most frequent keywords derived from Noun Phrases (NPs) in the input SRS. Then a query engine is implemented to query these keywords in Wikipedia. The output is a corpus consisting of Wikipedia articles whose titles overlap with any of the extracted N keywords. We then filter these articles and keep only matching articles that have high similarity to the original SRS. The intuition of this filtering is to avoid extracting out-of-domain articles. The corpus is passed on to step 4.

5.3.4 Document Retrieval

This step relies on IR-based methods to retrieve from the corpus the top relevant article to q. Specifically, we apply a *Retriever* model (*Retriever*, for short) that takes as input one article at a time paired with q. For each article in the corpus, we calculate the relevance between the article and q. This results in a score between 0 and 1, with 1 indicating identical. We then rank the articles in descending order according to their similarity scores against q and select the most relevant. As we explain in further detail in Section 5.4.4, we experiment with several *Retrievers* that are widely used in the QA literature [188]. The output of this step is the most relevant article to q.

5.3.5 Context Retrieval

This step is similar to step 4. The only difference is that the *Retriever* computes the relevance score between the contexts of the input SRS identified in step 1, one at at time, against q. If q is predicted as DomQ, then this step is performed on the most relevant article from the previous step. We note that the article is also passed through the same preprocessing step (step 1 in Fig. 5.2). Following this, the output of this step is the set of top-K relevant contexts to q.

5.3.6 Answer Extraction

The last step in our approach is to extract the answer to q from the top-K retrieved contexts. To do so, we use a *Reader* model (*Reader*, for short) that takes both q and a context as input and returns a text span in the context that is likely to provide the answer to q. As we explain in Section 5.4.4, we experiment with several *Readers* which are widely known to have good performance on different QA datasets [206].

The overall output of our approach is a likely answer highlighted in each of the top-K selected contexts. This way, we provide the human analyst with the necessary context to understand and validate the extracted answers. We discuss different values of K in Section 5.4 and the impact of selecting K in practice.

5.4 Empirical Evaluation

In this section, we empirically evaluation our *REQAssis* approach.

5.4.1 Research Questions (RQs)

Our evaluation addresses the following RQs:

RQ1: Which ML classification algorithm is the most accurate for distinguishing between document-based and domain-based questions? Step 2 in our approach (i.e., *Question Analysis*) identifies the question type as discussed in Section 5.3. RQ1 investigates alternative classification algorithms and selects the one that yields the most accurate results.

RQ2: Which Retriever model yields the most accurate results for retrieving for a given question the most relevant (a) domain-based article and (b) the most relevant contexts within a document? Steps 4 and 5 in our approach (i.e., *Document Retrieval* and *Context Retrieval*) can be implemented using several alternative *Retrievers*. Specifically, Step 4 applies the *Retriever* to retrieve from the domain-based corpus a Wikipedia article that likely contains the answer to the input question. Then, step 5 applies the *Retriever* to select from the input SRS (in case of DocQ) or from the most relevant article identified in step 4 (in case of DomQ) the top-K relevant contexts that likely contain the answer. Compared to step 4, step 5 provides a more-focused view to better pinpoint the answer to the input question. RQ2 identifies the most accurate alternative for each step. The most accurate *Retrievers* are used the to answer the subsequent RQs.

RQ3: Which Reader model yields the most accurate results for extracting the likely answer to a given question? Step 6 in our approach (i.e., *Answer Extraction*) delineates the likely answer in the top-K relevant contexts to a given question. In RQ3, we experiment with multiple alternative models and identify the most accurate alternative.

RQ4: Does our approach run within practical time? RQ4 analyzes the execution time required by our approach to analyze a given question, provide the relevant contexts and highlight the likely answer. For our approach to be applicable in practice, the overall pipeline described in Section 5.3 has to be performed in practical time to improve the efficiency of manual inspection of a given SRS.

5.4.2 Implementation Details

We implemented our approach (in Fig. 5.2) in Python 3.7.13. We implemented step 1 (Preprocessing) using the following libraries: For parsing the text of the input SRS (provided as MS Word document), we used Python-docx 0.8.11 library[1]. For operationalizing the NLP pipeline, we employed the T5 tokenizer from the Transformers 3.0.1 library [170], Porter and Snowball Stemmers, English stopwords list, Word tokenizer, WordNet, and WordNet Lemmatizer available in NLTK 3.2.5 [127], Python RE 2.2.1 regex library[2], and Tokenizer, Dependency Parser, and Entity Recognizer from SpaCy 3.3.0 [128]. For implementing step 3

[1] `https://github.com/python-openxml/python-docx`
[2] `https://docs.python.org/3/library/re.html`

(*Question Analysis*), we used the following libraries. Scikit-learn version 1.0.2 [130], for ML modeling including test-train splitting, TF-IDF transformation, training and testing of ML models, cross-validation, hyperparameter tuning, scoring and evaluation. For domain-specific corpus generation, in addition to the preprocessing libraries, we used Wikipedia library, version 1.4.0[3]. For both *Document Retrieval* and *Context Retrieval*, in addition to NLTK, Transformers, and Scikit-learn, we used ElasticSearch version 7.9.1 [207], Beir version 1.0.0 [208], and Rank-BM25 version 0.2.2 [209]. For *Answer Extraction*, in addition to SpaCy and Scikit-learn, we used Haystack version 1.4.1rc0 [210], and PyTorch version 1.11.0+cu113 [171]. The approach and experiments in this chapter are conducted and made available using Jupyter Notebooks [137].

5.4.3 Data Collection Procedure

Our data collection procedure aimed at collecting questions and answers from industrial SRSs covering diverse domains. To that end, we collected six SRSs covering three domains, namely aerospace, defence, and security. Despite the availability of several QA datasets, none of them is fitting the RE application discussed in our work. Hence, we created the *REQuestA* (standing for RE Question-Answering dataset). Further and to reduce the cost of the annotation process, about 50% of the question-answer pairs in REQuestA are generated fully automatically and then validated by human analysts as we discuss later in this section. Next, we discuss the desiderata specific to REQuestA, our automatic QA generation method, the annotation process, and finally our ground truth.

Desiderata. To ensure that REQuestA enables the development of a QA assistant for requirements engineers, we posit the following specific desiderata:

(1) *Focusing on content-based questions*. Existing datasets in RE that cover questions related to RE tasks, e.g., "which requirements are affected if I change this requirement". Such a question can be useful for change impact analysis. In contrast, REQuestA focuses on content-based clarification questions concerning quality issues.

(2) *Ensuring distance between questions and answers*. To provide meaningful assistance, the answer to a question posed by a requirements engineer on a particular requirement is not expected to be in that requirement or its surrounding context. Instead, the answer is expected to be in distant parts of the same SRS, i.e., outside the range of the engineer's current review. The motivation behind this desideratum is to (i) differentiate REQuestA from a conventional MRC dataset (see Section 5.2) which is not useful for requirements engineer, and (ii) emphasize the RE application scenario of our approach, i.e., assisting engineers in their review activity.

(3) *Enabling the acquisition of domain knowledge*. A requirements engineer should be able to pose questions related to domain knowledge. In our work, we motivate this desideratum with the possibility of generating glossary terms via templated questions, e.g., *"what is the definition of [concept]$_x$"*. To provide a more generic view, we also consider other domain-based clarification questions.

QA Auto-generation. To better provide clarification, REQuestA contains both *document-based questions (DocQ)* whose answers can be extracted from the input SRS, and *domain-based questions (DomQ)* whose answers are obtained by mining an external domain-specific knowledge resource. REQuestA further covers diverse types of questions based on the expected answer, including yes/no, short span answer, and sentence answer. We refer to a question-answer pair as $\langle q_c, a_c \rangle$ (in case of DocQ) and $\langle q_m, a_m \rangle$ (in case of DomQ). Fig. 5.3 shows the overview of our auto-generation method. Given an SRS as input, our method generates a list of $\langle q, a \rangle$ pairs in five steps, elaborated next.

(1) *Preprocessing*: In this step, we preprocess the input SRS using the NLP pipeline discussed in Section 5.3 extended with three more modules: *part-of-speech (POS) tagging* assigns a POS to each token, *text chunking*

[3] https://pypi.org/project/wikipedia/

groups words that act as one syntactic unit, e.g., noun phrases (NP), and *stopwords removal* eliminates very frequent words, e.g. prepositions. This step results in a preprocessed SRS that is passed on the next two steps.

(2) *Analyzing Domain*: Inspired by [10], we identify in this step the domain-specific concepts which can trigger a domain-based question. We do so by computing for each NP occurring in the SRS a variant of TF-IDF score. The TF is estimated from the input SRS and the IDF is computed considering the termsn in the SRSs of the other domains. Further and to ensure domain specificity, we filter out NPs that are contained in WordNet [98], a generic lexical database for English. The intuition is that if an NP is included in WordNet, then it might be a generic word/phrasewhich is falsely assigned a high TF-IDF score, e.g., "Lunar Rover". We then sort the NPs in descending order and select the top-50, referring to them as *keywords*. In a similar manner to generating domain-specific corpus proposed in the literature [10], we use each keyword to query Wikipedia and find a matching article, i.e., an article whose title overlaps with the keywords. For quality purposes, we generate $\langle q, a \rangle$ from Wikipedia articles only when they have semantic relatedness to the input SRS greater than a threshold. In our work, we set threshold value to 0.5. Greater and lower threshold values produce articles that are very similar or very dissimilar in content to the input SRS, and thus less informative either way for answering clarification questions (the goal of our approach). The result of this step is a set of Wikipedia articles which are closely related to the input SRS, and the set of seed keywords that were used to find these articles.

(3) *Partitioning*: In this step, we automatically partition a document into a set of contexts using a fixed-size sliding window [131]. Our partitioning aims at satisfying two objectives. First, the context size should not exceed 512 tokens, a constraint imposed by the architectures of QA technologies. Second, a context should be coherent, i.e., preserves an idea. To meet these objectives, we define a *context* as a set of adjacent sentences including an overlapping sentence with the preceding and/or following context. This procedure is ignored in case of paragraphs that do not exceed the token limit, as we believe a defined paragraph can act as a stand-alone context. To generate $\langle q_c, a_c \rangle$, we partition the input SRS. To ensure the above second desideratum, we filter out *isolated contexts*, i.e., such contexts that have zero relatedness to all other contexts in SRS. To generate $\langle q_m, a_m \rangle$, we partition each Wikipedia article resulted from step (2) explained above.

(4) *Generating* $\langle q, a \rangle$: In this step, we feed in to a question generation (QG) model each context from step (3) and an answer type for the desired question. The QG model first extracts an answer according to the answer type and then automatically generates a respective question from the given context. In our work, we apply three models based on T5 for generating the different questions, except for definition questions for which we apply a predefined template using the keywords extracted in step (2) as concepts. For these definition questions, we feed in both the context and the question to a QA model that extracts the answer. This model is again based on T5. Note that for validity reasons, *none of the models* contributed to generating our dataset are experimented with in Section 5.4. The output of this step is then the tuple $\langle d, c, q, a \rangle$, where d and c are the document and context containing the answer a to the question q. The first element d is the input SRS in case of $\langle q_c, a_c \rangle$, and a Wikipedia article in case of $\langle q_m, a_m \rangle$.

(5) *Evaluating meaningfulness of* $\langle q, a \rangle$: In the last step, we evaluate the meaningfulness of each auto-generated $\langle q, a \rangle$. In particular, we utilize the model *BERT-base-uncased-QA evaluator*, which computes a score for a given $\langle q, a \rangle$ based on a large QA training corpus exposed to the model. Finally, we select the top-N $\langle q, a \rangle$ ranked to be meaningful.

Annotation. The annotation process involved two human analysts, each having more than three years of experience in annotation tasks in the context of RE. The first analyst has a Master's degree in multilingualism and she did a three-month internship about ambiguity in requirements. The second analyst has a Masters degree

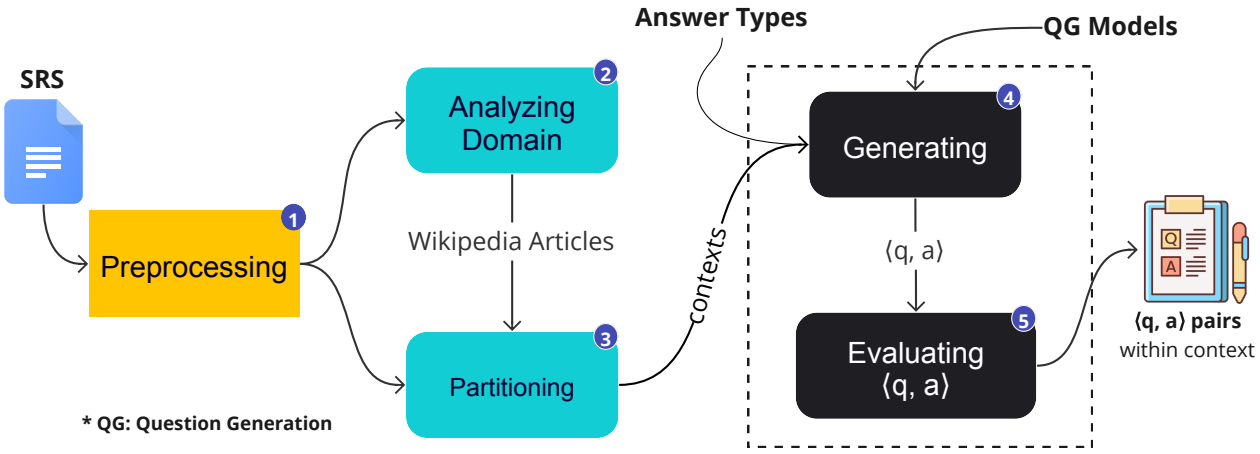


Figure 5.3: Overview of our QA Auto-Generation Method.

in quality assurance over computer science background. Both human analysts underwent half-day training session on question-answering with a focus on RE.

We shared with the annotators, the original SRSs and the lists of automatically generated tuples $\langle c, q, a \rangle$ for each SRS and domain considered in our analysis. The annotators were then asked to examine the tuples as follows. Except for templated questions which are assumed to be correct, each auto-generated question q was labeled as *valid* indicating that q was correct as-is, *rephrased* indicating that q was semantically correct but required structural improvement to become valid, or *invalid* indicating that q did not make much sense. Similarly, each auto-generated answer a was labeled as *correct*, *corrected*, or *incorrect* with similar indications to the ones mentioned above for q. Additionally, a could be labeled as *not in context*, when the answer to the question cannot be extracted from the context (we considered these cases as invalid). We further asked the annotators to manually add question-answer pairs on each context c (if possible). For domain-based questions, they were recommended to use external resources such as Wikipedia as reference to broaden their options for providing questions related to the domain of the context.

To construct the *REQuestA* dataset, we filtered out any tuple if q or a was labeled as invalid or incorrect. For the remaining tuples, we replaced the auto-generated q and a with the rephrased q and corrected a by the human annotator, respectively. We further appended the human-based q and a and the respective c. In total, *REQuestA* contains a list of 387 tuples in the form of $\langle c, q, a \rangle$ split by input SRS, and by domain. Table 5.1 reports the results of our document collection. Specifically, the table shows the total number of sentences $\mathcal{S}$ for each SRS, the total number of contexts in each SRS c_c, and the average number of contexts from the domain-specific articles considered per question in each SRS. The table then lists for each SRS both DocQ and DomQ, the number of auto-generated question-answer pairs $\langle q_c, a_c \rangle$ and $\langle q_m, a_m \rangle$, as well as the number of question-answer pairs provided by annotators h_c and h_m, respectively. The table also reports for each domain the size of the domain-specific corpus $\mathcal{C}$ computed as the total number of Wikipedia articles in the corpus.

In summary, we generated a total of 204 question-answer pairs, of which 111 are document-based and 93 are domain-based. We introduced 69 (of 93) templated questions for the purpose of glossary extraction from external domain-specific resources. We filtered out 31 tuples from *REQuestA* due to invalid questions and/or answers. The remaining tuples, a total of 173 (86 DocQ and 87 DomQ questions), alongside the human-based questions, a total of 214 (103 DocQ and 111 DomQ questions), constitute our ground truth. We note that answers can appear in more than one context due to the nature of the requirements and the employed context definition technique.

Table 5.1: Results of Document Collection.

Domain	$\mathcal{C}$†	SRS	$\mathcal{S}$‡	c_c	$\langle q_c, a_c \rangle$	h_c¶	c_m	$\langle q_m, a_m \rangle$	h_m¶
Aerospace	1158	SRS1	202	24	8	18	19	8	15
		SRS2	1500	107	37	40	23	37	38
Defence	747	SRS3	112	11	5	4	67	5	12
		SRS4	782	71	19	26	27	33	38
Security	46	SRS5	110	18	15	13	23	4	8
		SRS6	32	4	2	2	-	-	-
Total	1951	6	2738	235	86	103	159	87	111

† $\mathcal{C}$ represents the size of the domain-specific corpus computed as the total number of Wikipedia articles.
‡ $\mathcal{S}$ represents the size of the SRS computed as the total number of sentences in the SRS.
‡ c_c represents the number of contexts in the SRS, and c_m represents the average number of contexts from the articles considered in the respective domain.
§ $\langle q_c, a_c \rangle$ and $\langle q_m, a_m \rangle$ represent the automatically generated document- and domain-based questions and answers, respectively.
¶ h_c and h_m represent the respective document- and domain-based question-answer pairs provided by human annotators.

Quality of REQuestA. As a quality measure, the two annotators annotate an overlapping subset equals to 10% of the auto-generated tuples. We consider that the annotators agreed when they select the same label for a given question or answer (i.e., valid or invalid). Note that valid questions include rephrased ones. On this subset, the annotators fully agreed on the labels of the questions. Out of 173 auto-generated questions, the two annotators rephrased 24 questions (representing ≈14%) and corrected 46 answers (representing ≈26%). To better assess the quality of our generation method, we further compare the generated questions to the rephrased ones by the annotators. Following best practices in the natural language generation literature and machine translation (MT) [131], we apply the following metrics: BLEU, ROUGE, METEOR, and BERTScore. The first three are well-known n-gram metrics that measure the overlap between the generated questions (q_g) and the rephrased ones (q_h). BLEU and METEOR compute the overlap with respect to the total number of tokens in q_g, while ROUGE computes the overlap with respect to the longest common sequence between q_g and q_h. The last metric measures the semantic equivalence between q_g and q_h. The resulting scores are as follows: BLEU=53.6%, ROUGE=31%, METEOR=39.8%, BERTScore=95.3%. These values indicate that q_g and q_h are very close semantically yet different at the lexical level to some extent. This implies that the automatic generation could successfully produce semantically correct questions, but failed in structuring them well.

In addition, we investigated the annotated answers. Out of 173 answers validated by the annotators, 47 answers were corrected (27%). Out of these corrected answers, about 57% subsumed the original extracted answers. In other words, the annotators expanded the answers to include missing tokens, e.g., the auto-extracted answer "software code" was corrected to "implemented software code".

5.4.4 Evaluation Procedure

To answer our RQs, we conduct the experiments explained below. We note that we perform these experiments independently in order to evaluate each step of our approach separately.

EXPI. This experiment answers RQ1. In EXPI, we evaluate widely used and recommended algorithms [211, 173, 11]. Specifically, we compare six ML algorithms, including Random Forest (RF), Decision Tree (DT), AdaBoost

(ADA), Logistic Regression (LR), Gradient Boosting (XGB), and Support Vector Machine (SVM). Each model is explained in detail in Chapter 2. Following best practices [212], we train the classifiers on feature embeddings (FE). The questions are first transformed into mathematical representations using two methods, the traditional TF-IDF method, and more advanced embeddings extracted from word2vec and Sentence BERT (SBERT), since they are reported to have good performance in text classification [11]. In this experiment, we compute the TF-IDF by considering each question as a document. We train the classifiers and tune the hyperparameters on 80% of REQuestA. We then evaluate the classifiers and report the results in **RQ1** on the remaining 20% of REQuestA. To compare the performance of the classifiers, we use *Accuracy (A)* computed as the number of correctly classified questions divided by the total number of questions attempted for classification.

EXPII. This experiment answers **RQ2**. In EXPII, we evaluate three widely used alternatives *Retrievers* explained in Section 5.2. Specifically, we experiment with two traditional sparse *Retrievers*, namely *TF-IDF* with cosine similarity and *BM25*. We further experiment with dense and reranking *Retrievers*, which are reported as the best performing models in BEIR benchmark [188]. These models are DistilBERT and BM25 combined with cross encoder. In our work, we restrict the re-ranking to top-10 results.

In EXPII, we evaluate the *Retrievers* applied in in steps 4 and 5 of our approach (see Fig. 5.2) to retrieve for a given question the most relevant article and the top-K relevant contexts. EXPII is divided accordingly into two sub-experiments elaborated next.

- **EXPII-a.** *Document Retrieval:* We compute the traditional TF-IDF representation as explained in Section 5.2 where the articles in the domain-based corpus together with the input question constitute the overall vocabulary. The question is regarded as a single document. For conducting EXPII-a, the *Retriever* selects the candidate article from the domain-based corpus
- **EXPII-b.** *Context Retrieval:* We compute the TF-IDF representation where the IDF is computed from each context within a document. Similarly, the input question is regarded as a single document. The question and the set of contexts constitute the overall vocabulary. For conducting EXPII-b, the *Retriever* selects the candidate context from the list of contexts in the input SRS or in the selected article from EXPII-a.

To compare the performance of the alternative *Retriever* models in EXPII, we apply common evaluation metrics in the IR literature [129] explained next. *Recall@K (R@K)* assesses whether the document (or context) containing the correct answer to the input question is found anywhere in the ranked list produced by the *Retriever* models. *Discounted cumulative gain@K (NDCG@K)* is similar to *R@K* except that it takes into account the rank given to the context that contains the right answer to the input question. *Mean Reciprocal Rank (MRR)* evaluates the rank given to the first relevant retrieved context, this metric disregards other relevant items in the same ranked list. *Mean Average Precision@K (MAP@K)* computes the precision relative to the rank of the relevant context in the list of K retrieved contexts, e.g., MAP is 1/3 when the relevant context is ranked 3.

In our work, MRR and MAP are equivalent since there is for each question one relevant context only according to our ground truth. We still report both similar to the common practice in the QA literature [185]. We further note that in EXPII-a we are interested in the most relevant article and this article is known in our ground truth. Therefore, we only report the conventional recall which similar to the definition of R@1 (explained above).

EXPIII. This experiment answers **RQ3**. To extract the answer to a given question, we apply several *Readers* that are reported to perform well in the QA literature [188]. Specifically, we apply *ALBERT*, *BERT*, *ELECTRA*,

Table 5.2: Accuracy of Classification Models for *Question Analysis* (**RQ1**).

	RF	DT	ADA	XGB	LR	SVM
TF-IDF	79.5	80.8	**87.2**	83.3	84.6	52.6
word2vec	**78.2**	56.4	71.8	70.5	52.6	52.6
SBERT	**84.6**	69.2	80.8	69.2	80.8	83.3

MiniLM, and *RoBERTa* (see Section 5.2 for more details on these models). To compare the performance of these *Readers*, we compute the *Accuracy (A)* as the number of correctly predicted answers divided by the total number of predicted answers found by the model. To decide whether a predicted answer (a_p) is correct or not, we compare a_p with the answer provided in the ground truth (a_{gt}) using three modes explained next.

1. *Exact matching (EM)*: a_p is considered correct only if it fully matches a_{gt}, otherwise a_p is incorrect.

2. *Partial matching (PM)*: a_p is correct if there is some overlap between a_p and a_{gt}, otherwise a_p is incorrect.

3. *Semantic matching (SM)*: For this mode, we first compute semantic similarity between a_p and a_{gt}. Following this, a_p is correct if the similarity score is greater than a threshold (in our case, we use 0.5). Otherwise, a_p is incorrect.

In addition to reporting accuracy, we also report F1-measure, another commonly reported metric in the QA literature. F1-measure is the harmonic mean computed as 2*P*R/(P+R), where P is the precision computed as the ratio of the overlapping tokens between a_p and a_{gt} to the total number of tokens in a_p, and R is the recall computed as the ratio of the overlapping tokens between a_p and a_{gt} to the total number of tokens a_{gt}. We then report the overall average of the F1-score for all questions.

EXPIV. This experiment answers **RQ4**. We report the execution time needed to run our QA assistant approach *REQassis* with the most accurate models from previous experiments. EXPIV is conducted on a Google Colaboratory cloud using the free plan with the following specifications: Intel(R) Xeon(R) CPU@2.20GHz, Tesla T4 GPU, and 13GB RAM.

5.4.5 Answers to the RQs

RQ1. *Which ML classification algorithm is the most accurate for distinguishing between document-based and domain-based questions?* Table 5.2 shows the accuracy (A) of the six alternative ML classification algorithms using different learning features. We highlight in the table the best accuracy achieved for each representation technique. As the table shows, RF performs well when it is trained over embeddings from both SBERT and word2vec. However, ADA outperforms RF when trained over TF-IDF vectors, as it has the best accuracy in this table. LR achieves good accuracy combined with TF-IDF and SBERT representation techniques, but its performance degrades drastically when trained over word2vec embeddings. On the other hand, both ADA and RF have stable average performance in all three representations. We further experiment with ensembling methods. As per our results, the best ensembling is using stacking, that is a procedure where a classifier learns to combine the predictions of single classifiers. Stacking ADA and XGB both trained over TF-IDF vectors, LR and SVM over SBERT embeddings, and RF trained over word2vec yields **88.5%** accuracy with a gain of 1.3% over the best classifier (ADA over TF-IDF vectors). This observation leads to the conclusion that the stacking solution performs best.

The answer to **RQ1** is that *stacking different Frequency-based, Embedding-based$_w$ and Embedding-based$_s$ models is the best performing classification approach* for distinguishing the two question types *with an accuracy of ≈88.5%*.

For the purpose of our study, we believe that the achieved accuracy of 88.5% is sufficient. We note that there can be alternatives to using ML classification for analyzing the question type. Considering the application use of our approach for extracting glossary terms from external domain knowledge resource, this classification task can be easily handled with nearly a perfect accuracy using heuristics about the question template. An alternative is to allow the user to provide the question type as an input parameter. Another alternative is to disregard the question type and provide two answers, one extracted from the input SRS (i.e., regarding the question as document-based) and another extracted from the domain-specific corpus (i.e., regarding the question as domain-based). This latter alternative requires more time to process a given question, yet it has the advantage of providing the requirements engineer with a domain-specific perspective of the input question.

RQ2. *Which Retriever model yields the most accurate results for retrieving for a given question the most relevant (a) domain-based document(s) and (b) the most relevant contexts within a document?* To answer (a), Table 5.3 shows the performance of the different *Retriever* models for retrieving the correct domain-based article. In the table, we assume that the DomQ questions are correctly classified, independently of the *Question Analysis* component discussed in the previous **RQ**. Lexical models (TF-IDF and BM25) can easily identify the correct article achieving a perfect Recall@1 of 100% over all domains. The dense *Retriever* (DistilBERT) has a slightly lower performance compared to the lexical approaches, achieving an average Recall@1 of 98.8%. Since the reranking approach uses the results of BM25, it achieves the same 100% Recall@1. Since three approaches achieve the same results, we select the BM25 model as the best *Retriever* for this *Document Retrieval* component since it is computationally efficient, simple and has a robust behavior. We note that the average reported in table 5.3 is calculated by weighting the domains by their corpus size.

To answer (b), Tables 5.4 and 5.5 show the performance of the different *Retrievers* for retrieving the top-K relevant contexts for DocQ and DomQ, respectively. In these tables, we assume that the DocQ and DomQ questions are correctly classified, and the correct article is retrieved for DomQ, independently of the results of the *Question Analysis* and *Document Retrieval* components discussed in **RQ1** and the first part of **RQ2**. For DocQ, the lexical models (TF-IDF and BM25) have a comparable performance, with BM25 having a slight edge over TF-IDF in the smaller values of K. The dense *Retriever* improves the results of BM25 reaching an average of ≈84% for R@3. Reranking *Retriever* which pairs BM25 with the CE model achieves the best overall results, that is an average of ≈85.8% for R@3. For DomQ questions, TF-IDF outperforms BM25 in most of the cases. However, the dense *Retriever* achieves better R@1 and R@3 than both lexical approaches. The reranking approach has a better overall Recall for all K values, achieving an average of ≈95.4% for R@3.

We observe from the results that the lexical approaches (BM25 and TF-IDF) perform better for DocQ than for DomQ, whereas the neural-based approaches (dense and reranking *Retrievers*) perform better for DomQ than for DocQ. We hypothesize that the reason can be due to the fact that DocQ contexts are extracted from the original input SRS on from where the question was originally asked/generated, while DomQ contexts are from Wikipedia articles. This might have an impact on the relevance score that is computed by the *Retrievers* to identify the right context.

The table further shows that selecting the value $K = 3$ yields good results on average for both DocQ and DomQ. In comparison, the results at $K = 1$ are often insufficient and miss the right context to a given question.

Table 5.3: Accuracy of *Retriever* Models for *Document Retrieval* (**RQ2-a**).

Domain	$\mathcal{C}$†	Model	Recall@1
Aerospace	1158	TFIDF	**100**
		BM25	**100**
		DistilBERT	99.0
		BM25+CE	**100**
Defence	781	TFIDF	**100**
		BM25	**100**
		DistilBERT	98.9
		BM25+CE	**100**
Security	50	TFIDF	**100**
		BM25	**100**
		DistilBERT	91.7
		BM25+CE	**100**
Average	663	TFIDF	**100**
		BM25	**100**
		DistilBERT	98.8
		BM25+CE	**100**

† $\mathcal{C}$ the domain-specific corpus size in numbers of Wikipedia articles.

Considering larger values of K ($K \geq 5$) is also possible and it improves the overall results. Selecting the best value of K has practical implications. In practice, selecting higher values of K provide the requirements engineer with more context required for understanding and interpreting the requirements. However, this comes at the cost of more time needed to navigate through the contexts and check the extracted answers. We therefore select $K = 3$ in our work, but we believe that K can be left as an external parameter to our approach and can be then adjusted by the requirements engineer according to the practical context.

The answer to **RQ2** is that BM25 is the best model for *Document Retrieval* with a perfect 100% R@1, and the reranking approach which pairs BM25 with CE performs the best for *Context Retrieval* with an average of R@3 90.6% .

RQ3. Table 5.7 compares the accuracy of the *Readers* for extracting the answer to a given question. In this table, we assume that the previous steps (**RQ2**) provided the context (and the Wikipedia article, when applicable) containing the right answer to the input question. The table shows that the most accurate *Reader* varies according to the matching mode. Consequently, the best performing model is DistilBERT in terms of PM mode, but RoBERTa has the best performance in terms of EM mode and also considering F1 score, and finally ALBERT has a slightly better accuracy in terms of SM mode mode than RoBERTa. Thus, we can argue that RoBERTa model always provides the best answer compared to other models, despite having a low PM score. MiniLM and DistilBERT are the most efficient models computational-wise, which makes them preferable in low-resource case studies, particularly DistilBERT, since it performs well in terms of PM. Though in PM mode the model does not identify the exact answer as-is in the ground truth, the good results of DistilBERT indicate that it still can point the engineer to the text fragment containing the right answer. ALBERT is the largest model that we use in this list, since it is based on BERT-large. Though ALBERT has comparable results to RoBERTa (even has superiority in terms of PM and SM), it is computationally expensive and thus less preferred than RoBERTa

Table 5.4: Accuracy of *Retriever* Models for *Context Retrieval* for DocQ (**RQ2-b**).

	SRS († c)	SRS1 (34)				SRS2 (123)			
	Model	TFIDF	BM25	DistilBERT	BM25+CE	TFIDF	BM25	DistilBERT	BM25+CE
Top-1	R	50.0	50.0	53.8	**80.8**	46.8	51.9	59.7	**75.3**
	MAP	50.0	50.0	53.8	80.8	46.8	51.9	59.7	75.3
	MRR	50.0	50.0	61.5	80.8	48.1	51.9	62.3	75.3
	NDCG	50.0	50.0	53.8	80.8	46.8	51.9	59.7	75.3
Top-3	R	73.1	76.9	84.6	**92.3**	66.2	66.2	83.1	**84.4**
	MAP	60.3	61.5	68.6	85.3	55.2	57.6	70.3	79.4
	MRR	60.3	61.5	72.4	85.3	56.1	57.6	71.9	79.4
	NDCG	63.6	65.5	72.8	87.0	58.0	59.8	73.6	80.7
Top-5	R	73.1	88.5	**96.2**	92.3	76.6	75.3	**89.6**	85.7
	MAP	60.3	64.0	71.5	85.3	57.7	59.7	71.8	79.7
	MRR	60.3	64.0	75.3	85.3	58.2	59.7	73.1	79.7
	NDCG	63.6	70.1	77.7	87.0	62.4	63.6	76.3	81.2
Top-10	R	84.6	92.3	**100**	92.3	85.7	87.0	**93.5**	87.0
	MAP	61.8	64.6	72.0	85.3	58.9	61.2	72.3	79.9
	MRR	60.9	64.6	75.9	85.3	59.4	61.2	73.8	79.9
	NDCG	67.2	71.4	79.0	87.0	65.3	67.3	77.6	81.7
	SRS († c)	SRS3 (13)				SRS4 (87)			
Top-1	R	44.4	44.4	22.2	**55.6**	48.9	55.6	**71.1**	68.9
	MAP	44.4	44.4	22.2	55.6	48.9	55.6	71.1	68.9
	MRR	44.4	44.4	22.2	55.6	48.9	55.6	71.1	68.9
	NDCG	44.4	44.4	22.2	55.6	48.9	55.6	71.1	68.9
Top-3	R	**88.9**	66.7	**88.9**	**88.9**	57.8	75.6	**84.4**	82.2
	MAP	64.8	53.7	55.6	72.2	51.9	64.8	77.8	75.6
	MRR	64.8	53.7	55.6	72.2	51.1	64.8	77.8	75.6
	NDCG	71.0	57.0	64.3	76.6	53.3	67.6	79.5	77.3
Top-5	R	**100**	77.8	**100**	**100**	66.7	77.8	**86.7**	82.2
	MAP	67.6	55.9	58.3	75.0	54.1	65.3	78.3	75.6
	MRR	67.6	55.9	58.3	75.0	53.3	65.3	78.3	75.6
	NDCG	75.8	61.3	69.1	81.4	57.2	68.5	80.5	77.3
Top-10	R	**100**	**100**	**100**	**100**	75.6	82.2	**91.1**	82.2
	MAP	67.6	58.9	58.3	75.0	55.3	65.8	78.9	75.6
	MRR	67.6	58.9	58.3	75.0	54.2	65.8	78.9	75.6
	NDCG	75.8	68.5	69.1	81.4	60.0	69.8	81.9	77.3
	SRS († c)	SRS5 (23)				SRS6 (4)			
Top-1	R	71.4	75.0	60.7	**92.9**	**100**	**100**	50.0	**100**
	MAP	71.4	75.0	60.7	92.9	100	100	50.0	100
	MRR	71.4	75.0	60.7	92.9	100	100	50.0	100
	NDCG	71.4	75.0	60.7	92.9	100	100	50.0	100
Top-3	R	85.7	85.7	**96.4**	92.9	**100**	**100**	50.0	**100**
	MAP	77.4	79.8	76.2	92.9	100	100	50.0	100
	MRR	77.4	79.8	76.2	92.9	100	100	50.0	100
	NDCG	79.5	81.3	81.4	92.9	100	100	50.0	100
Top-5	R	89.3	92.9	**96.4**	92.9	**100**	**100**	50.0	**100**
	MAP	78.3	81.5	76.2	92.9	100	100	50.0	100
	MRR	78.3	81.5	76.2	92.9	100	100	50.0	100
	NDCG	81.0	84.4	81.4	92.9	100	100	50.0	100
Top-10	R	92.9	92.9	**96.4**	92.9	**100**	**100**	50.0	**100**
	MAP	78.9	81.5	76.2	92.9	100	100	50.0	100
	MRR	78.9	81.5	76.2	92.9	100	100	50.0	100
	NDCG	82.3	84.4	81.4	92.9	100	100	50.0	100

† c represents the number of contexts.

Table 5.5: Accuracy of *Retriever* Models for *Context Retrieval* for DomQ (**RQ2-b**).

(a) Aerospace Domain

	SRS († c)	SRS1 (19)				SRS2 (23)			
	Model	TFIDF	BM25	DistilBERT	BM25+CE	TFIDF	BM25	DistilBERT	BM25+CE
Top-1	R	43.8	56.2	78.1	**85.7**	43.4	44.7	55.3	**64.5**
	MAP	43.8	56.2	78.1	85.7	43.4	44.7	55.3	64.5
	MRR	39.0	100	78.1	76.2	40.8	100	53.9	59.2
	NDCG	43.8	56.2	78.1	85.7	43.4	44.7	55.3	64.5
Top-3	R	61.0	85.7	90.5	**95.2**	68.4	80.3	84.2	**94.7**
	MAP	51.1	69.7	84.3	90.5	54.8	61.6	69.1	78.7
	MRR	51.9	100	84.3	85.7	54.4	100	68.4	76.1
	NDCG	53.6	73.8	85.9	91.7	58.3	66.5	73.0	82.9
Top-5	R	61.0	**95.2**	90.5	**95.2**	76.3	88.2	90.8	**94.7**
	MAP	51.1	71.6	84.3	90.5	56.5	63.5	70.5	78.7
	MRR	51.9	100	84.3	85.7	55.6	100	69.9	76.1
	NDCG	53.6	77.5	85.9	91.7	61.4	69.7	75.7	82.9
Top-10	R	85.7	**95.2**	90.5	**95.2**	**97.4**	94.7	92.1	94.7
	MAP	54.5	71.6	84.3	90.5	59.2	64.3	70.7	78.7
	MRR	52.7	100	84.3	85.7	56.2	100	70.1	76.1
	NDCG	61.7	77.5	85.9	91.7	68.2	71.8	76.1	82.9

(b) Security Domain

	SRS († c)	SRS3 (67)				SRS4 (27)			
Top-1	R	41.7	44.4	83.3	**88.9**	42.1	31.6	**71.1**	63.2
	MAP	41.7	44.4	83.3	88.9	42.1	31.6	71.1	63.2
	MRR	52.8	100	83.3	88.9	28.9	100	69.7	46.1
	NDCG	41.7	44.4	83.3	88.9	42.1	31.6	71.1	63.2
Top-3	R	58.3	86.1	88.9	**94.4**	65.8	76.3	90.8	**94.7**
	MAP	49.1	63.9	86.1	91.7	53.5	52.4	79.6	78.5
	MRR	57.4	100	86.1	91.7	46.7	100	78.9	70.0
	NDCG	51.5	69.6	86.8	92.4	56.7	58.6	82.5	82.7
Top-5	R	58.3	**94.4**	88.9	**94.4**	75.0	84.2	93.4	**94.7**
	MAP	49.1	65.6	86.1	91.7	55.4	54.1	80.2	78.5
	MRR	57.4	100	86.1	91.7	47.5	100	79.5	70.0
	NDCG	51.5	72.9	86.8	92.4	60.3	61.8	83.5	82.7
Top-10	R	77.8	**94.4**	88.9	**94.4**	94.7	94.7	**96.1**	94.7
	MAP	51.9	65.6	86.1	91.7	57.9	55.6	80.5	78.5
	MRR	59.3	100	86.1	91.7	48.9	100	79.9	70.0
	NDCG	57.9	72.9	86.8	92.4	66.5	65.2	84.4	82.7

(c) Defence Domain

	SRS († c)	SRS5 (23)			
	Model	TFIDF	BM25	DistilBERT	BM25+CE
Top-1	R	60.0	66.7	63.3	**80.0**
	MAP	60.0	66.7	63.3	80.0
	MRR	0.0	100	43.3	36.7
	NDCG	60.0	6.7	63.3	80.0
Top-3	R	80.0	73.3	**100**	**100**
	MAP	70.0	40.0	80.0	90.0
	MRR	33.3	100	70.0	68.3
	NDCG	72.6	48.7	85.2	92.6
Top-5	R	80.0	**100**	**100**	**100**
	MAP	70.0	45.7	80.0	90.0
	MRR	33.3	100	70.0	68.3
	NDCG	72.6	59.3	85.2	92.6
Top-10	R	**100**	**100**	**100**	**100**
	MAP	72.2	45.7	80.0	90.0
	MRR	34.3	100	70.0	68.3
	NDCG	78.7	59.3	85.2	92.6

† represents the average number of contexts.

Table 5.7: Accuracy of *Reader* Models for *Answer Extraction* (**RQ3**).

Model	Accuracy			F1	Time
	EM	PM	SM		
RoBERTa	**24.6**	60.2	84.0	**65.2**	11.6
BERT	21.4	70.6	82.9	63.1	32.2
DistilBERT	23.0	**86.4**	75.9	61.0	**5.8**
ALBERT	24.3	79.1	**84.2**	64.6	193.2
ELECTRA	21.1	81.3	81.0	60.1	19.1
MINILM	23.3	73.3	82.4	63.4	**5.0**

model. ELECTRA and MiniLM did not perform well on average compared to the other models.

The answer to **RQ3** is that RoBERTa is the best performing *Reader* with an accuracy of 84% indicating that the answers extracted by RoBERTa are semantically relevant to the right answers. RoBERTa further achieves the best average F1-score of 65.2%.

RQ4. To answer this RQ, we consider the best models from the previous RQs. Therefore, we consider the stacking approach for the *Question Analysis* step, reranking BM25 paired with the CE method for the *Document Retrieval* and *Context Retrieval* steps, and RoBERTa for the *Answer Extraction* step. Accordingly, we discuss next the execution time required to run *ReQAssis* from end-to-end estimated for a given question. Running the steps *Preprocessing* and *Question Analysis* requires 0.7 seconds. Then, running the *Document Retrieval* and *Context Retrieval* steps takes 4.9 and 3.5 seconds, respectively. We note that DomQ questions require both steps (i.e., 8.4 seconds), while DocQ questions require the *Context Retrieval* step only (3.5 seconds). In our analysis, DomQ constitute about half of the total questions in our dataset. However, we anticipate that requirements engineers pose less DomQ than DocQ in practice. The reason is that the engineers would have some familiarity with the underlying domain, and our automation is mainly intended for improved handling of quality issues, mainly addressed using DocQs.

Finally, the *Answer Extraction* step requires 0.1. The total execution time required on average to process a single question is approximately 4.3 seconds for DocQ questions and 9.2 seconds for DomQ questions. We note that we did not account in our estimation for the *Domain-specific Corpus Generation* step, since this step can be done offline. In the above estimation, we focused more on the use of *ReQAssis* during online inspection sessions.

The answer to **RQ4** is that *ReQAssis* requires 4.3 seconds to answer DocQ and 9.2 seconds to answer DomQ on a given SRS. Scaling this up to 50 questions, the total execution time will be around 3.5 min for DocQ, and 7.6 min for DomQ questions (5 min on average). From a practical standpoint, this can be considered convenient, as the practical use case involves multiple engineers inspecting the same SRS often for a duration of two hours. We believe that applying *ReQAssis* to clarify problematic requirements on the spot provides accurate results within practical time.

5.5 Related Work

In this section, we contrast our work with the existing literature on QA in the RE and NLP fields.

Question Answering in RE.

QA has been studied in the RE community with limited scope focusing on traceability [24, 25, 26]. Malviya et al. [213] investigate the potential questions and the artifacts that a requirements engineer typically asks. The authors collected by means of survey with industry practitioners a set of 159 queries, grouped into nine different categories and 53 sub- categories. These categories represent the RE activities for which the query is most relevant. Our work is closely related to the category *quality* proposed by the authors. However, unlike the questions that they address, we enable requirements engineers posing questions that are about the content of the requirements. For example, instead of asking "which requirements are incomplete?", the requirements engineer can pose the question "how distance is measured?" on a particular requirement. This way, if one requirement is incomplete on its own, the engineer is still able to find the complementary information in the SRS during inspection. Pruski et al. [25] present TiQi, an automated solution that transforms trace queries from natural language into structured query language (SQL). TiQi is designed to answer queries from a database, i.e., all data and traceability links have to be extracted in advance and stored in a SQL database. Compared to TiQi, our work focuses on answering questions posed in NL that arise while inspecting a given SRS concerning quality assurance activities. For that purpose, we limit extending the information in the input SRS to a domain-specific knowledge resources automatically generated throughout our work to better address quality questions.

Other directions for applying QA in RE literature include requirements elicitation [214], and information extraction form legal text [66]. The first direction provides guidelines in formulating questions that are useful to elicit a complete set of requirements, whereas the second direction aims at helping the requirements engineers better understand legal text that is relevant to requirements. For answering legal queries, the automation utilizes a set of pre-defined information content (e.g., prohibition or constraint). Compared to the above, our work (i) provides automated assistance that covers evaluating quality-related issues and acquiring domain-specific knowledge in requirements; (ii) relies merely on the unstructured textual content of SRS and (if needed) consults an external domain-specific knowledge resource.

In a wider context, QA is more recently investigated for chatbots in software engineering [215]. Zhang's team [216] introduce an interactive approach to improve retrieving questions and answers from technical platforms like Stack Overflow. Their approach builds a chatbot that interacts with and assists the user in finding more desired results. Bansal et al. [217] build a context-based QA system that covers basic programming questions about subroutines in programs (e.g., Java programs).

Question Answering in NLP. Question Answering (QA) is a long-standing topic in the NLP community. QA tasks include question classification, answer extraction, question-answer matching, knowledge base question answering, and question generation [218, 219, 220, 221]. Answer extraction is considered the main QA task in NLP, it involves extracting exact answers from related contexts or documents [222]. Recent advances in QA answer extraction include fine-tuning various state-of-the-art deep learning-based language models such as BERT, RoBERTa, and ALBERT [223, 224, 225, 226]. The question classification task involves determining the type of questions or answers, or the intent of the question in the case of chatbots and AI-based conversational systems [227]. Inspired by the NLP literature, we apply in our work the QA models that are reported in BEIR leaderboard, a recent benchmark for QA in NLP.

Several existing datasets are curated from generic text (e.g., Wikipedia) are publicly available for QA.

Among these, we mention the widely-used ones including SQuAD [228], GLUE [229], Natural Questions [230], MS MACRO [231], and TriviaQA [232]. These datasets are considered as benchmarks for the QA tasks. There are some existing domain-specific datasets covering various domains [233, 234, 235, 236] (e.g., medical domain [237]). None of the above-mentioned datasets is suitable for the purpose of our study in the context of QA assistance for requirements engineering. Available datasets either lack questions that capture domain-specific knowledge needed by requirements engineers, or do not explicitly cover the types of questions potentially posed by the engineer during inspection sessions (the focus of our work). Thus, we opted for creating an RE-specific dataset and making it available to the community.

The recent emerging of sequence-to-sequence language models enabled huge advances in text generation related tasks [238], such as dialog systems and Question Generation (QG). QG systems utilize the encoder-decoder structure and attention mechanisms (e.g., BART and T5 model) [239, 195, 204]. Such models enabled researchers in many fields to automatically generate their own synthetic QA datasets [240, 241, 242, 243]. In our work, we use QG models to generate our dataset.

Compared to the QA literature in NLP discussed above, our work differs in two ways. First, we provide an end-to-end textual question answering solution. Specifically, we devised our approach to cover all different steps starting from posing a question, all the way to providing the relevant contexts and potential answers. In contrast, the research directions in NLP often focus on one step only, e.g., document/context retrieval or answer extraction. Second, our work aims at providing automated assistance to requirements engineers at two levels, document-based questions that are posed and answered from the SRS under review, as well as domain-based questions that require mining external knowledge-based resources.

5.6 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 mainly impacts our dataset *REQuestA*. To mitigate this threat, the authors had no involvement in the annotation activity of *REQuestA*. Instead, the annotation was performed exclusively by two third-party nonauthor annotators who had no exposure to our implementation.

External Validity. Our evaluation is based on *REQuestA* dataset, which spans six industrial requirements specifications, covering three different domains. The results we obtained across these domains alongside the variety of question types considered in *REQuestA* provides confidence about generalizability of our empirical findings. Further experimentation is nevertheless required to further mitigate the external-validity threats.

5.7 Conclusion

In this chapter, we proposed an AI-enabled automated solution that uses question-answering (QA) as a way to support the requirements engineers in inspecting requirements. We call this solution *ReQAssis*, which stands for Quality Assurance Assistance through Question Answering. ReQAssis is composed of three main components: question analysis, information retrieval, and answer extraction. For each component, we used and compared different advanced machine learning (ML) and natural language processing (NLP) techniques. We empirically evaluated our solution on our semi-generated QA dataset *REQuestA*, which stands for Requirements Engineering Question-Answering dataset. *REQuestA* contains a total of 387 question-answer pairs of which 173

are automatically generated. The question analysis component could distinguish document-based and domain-based questions with 88.5% accuracy, using stacking to combine multiple ML models based on frequency and embeddings. The information retrieval component could retrieve the relevant document for domain-specific questions with 100% accuracy, and retrieve the text passage that contains the right answer to the input question among the top three relevant text passages with an average accuracy of 90.6%. The answer extraction component could extract from the right passage the likely answer to the question with an average accuracy of 84%. These results show that our solution can accurately provide three alternative answers to a document or a domain question, which can be useful in practice. This solution has some room for improvement, and it can evolve to an interactive chatbot.

Chapter 6

Conclusion

This chapter concludes this dissertation by summarizing the previous chapters, outlining the main contributions, and exploring future research directions.

6.1 Summary

In this dissertation, we presented multiple solutions to handle different types of ambiguity in software requirements specifications. We further proposed a question-answering-based requirements analysis assistance system based on advanced machine learning (ML), NLP, and information retrieval technologies. We empirically evaluated our automated solutions on RE-related datasets that we designed and got annotated by third-party annotators. We have implemented and are releasing under open-source licenses our tools with the best-performing solutions for handling ambiguity and QA-based assistance. Briefly, the contributions made in the different chapters of this dissertation are as follows:

In Chapter 3, we proposed an automated approach to improve the handling of two types of ambiguity, coordination ambiguity (CA) and prepositional-phrase attachment ambiguity (PAA) using a combination of syntactic patterns and different types of heuristics. We further proposed in Chapter 3 an automated module for extracting domain-specific corpora by crawling Wikipedia. We utilized this module to increase the accuracy of CA and PAA handling in requirements specifications. We evaluated our approach on a dataset with more than 5000 industrial requirements covering seven different application domains. Our results showed that our approach can detect CA and PAA with an average precision of $\approx$80% and an average recall of $\approx$89%. The results further showed that employing domain-specific corpora has a substantial positive impact on the accuracy of CA and PAA handling. Specifically, in our dataset, we observed a $\approx$33% improvement in accuracy when compared to a baseline that uses generic corpora.

In chapter 4, we tackled another type of ambiguity: pronominal anaphoric ambiguity. We proposed and developed six solution alternatives based on ML over language features and feature embeddings, fine-tuning of language models, and widely-applied NLP coreference resolution tools. Our evaluation involved two datasets, DAMIR, a requirements engineering anaphoric ambiguity dataset created for this task and annotated by external annotators, in addition to another publicly available RE dataset. Our results indicated that, for anaphoric

ambiguity detection, supervised ML is more accurate than alternatives, having an average precision of ≈60% and a recall of 100%. For anaphora resolution, the SpanBERT language model (a variant of BERT) yields the best performance with an average success rate of ≈98%.

In Chapter 5, we proposed ReQAssis, short for Requirements Quality Assurance Assistance through Question Answering. ReQAssis is an automated approach that uses question-answering (QA) to support the requirements inspection and analysis process. ReQAssis combined information retrieval and machine reading comprehension and further incorporates domain-specific knowledge to provide answers to the requirements engineers regarding quality issues in requirements (e.g., incomplete requirements, or unclear concepts). We evaluated our approach on REQuestA, a semi-automatically generated RE question answering dataset. Our approach achieved an accuracy of 88.5% for classifying a given question into a document-based (whose answer is in the requirements specification) or a domain-based question (whose answer requires mining an external domain-specific resource). Our approach further achieved 100% accuracy for extracting the relevant document that contains the answer from a given domain-specific corpus, an average accuracy of 90.6% in retrieving the text passage(s) that contain the likely answer to the input question among the top-3 text passages marked as relevant, and an accuracy of 84% in extracting the likely answer from the retrieved text passage(s).

6.2 Future work

In this dissertation, we have addressed three types of syntactic ambiguity in natural-language requirements. 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 by considering other under-studied ambiguity types in RE (e.g., semantic ambiguity). Transforming textual requirements specifications into structured information, combining our ambiguity handling solutions into one tool, and considering other potentially prevalent ambiguity types are potential future directions that can improve the effectiveness and usefulness of automation for requirements quality assurance.

Furthermore, our solution for coordination and prepositional-phrase attachment ambiguity can be extended by combining straightforward methods based on syntactic patterns and heuristics with more advanced technologies relying on language models and/or machine learning algorithms. In relation to our question-answering-based quality assurance system, an interesting avenue for future work is to build an interactive, user-friendly chatbot around the existing system. Our QA system can be further extended to embed our ambiguity handling solutions and consequently pinpoint problematic text passages (requirements in this case). Finally, more user studies are necessary to better assess the usefulness and applicability of our solutions and to more conclusively evaluate our current results in practical scenarios.

Bibliography

[1] Benedetta Rosadini, Alessio Ferrari, Gloria Gori, Alessandro Fantechi, Stefania Gnesi, Iacopo Trotta, and Stefano Bacherini. 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*, 2017.

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

[3] Klaus Pohl. *Requirements engineering: An overview*. RWTH, Fachgruppe Informatik Aachen, 1996.

[4] Pamela Zave. Classification of research efforts in requirements engineering. *ACM Computing Surveys (CSUR)*, 29(4):315–321, 1997.

[5] R Ryan Nelson. IT project management: Infamous failures, classic mistakes, and best practices. *MIS Quarterly executive*, 6(2), 2007.

[6] Axel Van Lamsweerde. *Requirements engineering: From system goals to UML models to software*, volume 10. Chichester, UK: John Wiley & Sons, 2009.

[7] Liping Zhao, Waad Alhoshan, Alessio Ferrari, Keletso J Letsholo, Muideen A Ajagbe, Erol-Valeriu Chioasca, and Riza T Batista-Navarro. Natural language processing for requirements engineering: a systematic mapping study. *ACM Computing Surveys (CSUR)*, 54(3):1–41, 2021.

[8] Klaus Pohl. *Requirements engineering: fundamentals, principles, and techniques*. Springer Publishing Company, Incorporated, 2010.

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

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

[11] Saad Ezzini, Sallam Abualhaija, Chetan Arora, and Mehrdad Sabetzadeh. Automated handling of anaphoric ambiguity in requirements: A multi-solution study. In *2022 IEEE/ACM 44th International Conference on Software Engineering*, 2022.

[12] Fabiano Dalpiaz, Ivor Schalk, and Garm Lucassen. Pinpointing ambiguity and incompleteness in requirements engineering via information visualization and NLP. In *Proceedings of the 24th Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ'18)*, 2018.

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

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

[15] Vincenzo Ambriola and Vincenzo Gervasi. On the systematic analysis of natural language requirements with CIRCE. *Automated Software Engineering*, 13(1), 2006.

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

[17] Chetan Arora, Mehrdad Sabetzadeh, Lionel Briand, and Frank Zimmer. Automated checking of conformance to requirements templates using natural language processing. *IEEE Transactions on Software Engineering*, 41(10), 2015.

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

[19] D. Berry, E. Kamsties, and M. Krieger. From contract drafting to software specification: Linguistic sources of ambiguity, a handbook, 2003.

[20] F. Chantree, B. Nuseibeh, A. de Roeck, and A. Willis. Identifying nocuous ambiguities in natural language requirements. In *Proceedings of the 14th IEEE International Requirements Engineering Conference*, 2006.

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

[22] Fabiano Dalpiaz, Davide Dell'Anna, Fatma Aydemir, and Sercan Cevikol. Requirements classification with interpretable machine learning and dependency parsing. In *Proceedings of the 27th IEEE International Requirements Engineering Conference*, 2019.

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

[24] Patrick Mäder and Jane Cleland-Huang. A visual language for modeling and executing traceability queries. *Software & Systems Modeling*, 12(3):537–553, 2013.

[25] Piotr Pruski, Sugandha Lohar, William Goss, Alexander Rasin, and Jane Cleland-Huang. Tiqi: answering unstructured natural language trace queries. *Requirements Engineering*, 20(3):215–232, 2015.

[26] Jinfeng Lin, Yalin Liu, Jin Guo, Jane Cleland-Huang, William Goss, Wenchuang Liu, Sugandha Lohar, Natawut Monaikul, and Alexander Rasin. Tiqi: A natural language interface for querying software project data. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 973–977. IEEE, 2017.

[27] Sallam Abualhaija, Chetan Arora, Amin Sleimi, and Lionel Briand. Automated question answering for improved understanding of compliance requirements: A multi-document study. In *In Proceedings of the 30th IEEE International Requirements Engineering Conference, Melbourne, Australia 15-19 August 2022*, 2022.

[28] Carson Schütze. PP attachment and argumenthood. *MIT working papers in linguistics*, 26(95), 1995.

[29] Paul Engelhardt and Fernanda Ferreira. Processing coordination ambiguity. *Language and Speech*, 53(4), 2010.

[30] Fabian de Bruijn and Hans Dekkers. Ambiguity in natural language software requirements: A case study. In *Proceedings of the 16th Working Conference on Requirements Engineering: Foundation for Software Quality*, 2010.

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

[32] Vincenzo Gervasi, Alessio Ferrari, Didar Zowghi, and Paola Spoletini. Ambiguity in requirements engineering: Towards a unifying framework. In *From Software Engineering to Formal Methods and Tools, and Back*. Springer, 2019.

[33] Saad Ezzini, Sallam Abualhaija, Chetan Arora, Mehrdad Sabetzadeh, and Lionel Briand. Maana: An automated tool for domain-specific handling of ambiguity. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings*, 2021.

[34] Saad Ezzini, Sallam Abualhaija, and Mehrdad Sabetzadeh. Wikidominer: Wikipedia domain-specific miner. *arXiv preprint arXiv:2206.10218*, 2022.

[35] Ruslan Mitkov. *Anaphora resolution*. Routledge, 2014.

[36] Saad Ezzini, Sallam Abualhaija, Chetan Arora, and Mehrdad Sabetzadeh. Taphsir: Towards anaphoric ambiguity detection and resolution in requirements. In *Proceedings of the 17th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (submitted)*, 2022.

[37] Christopher Manning and Hinrich Schütze. *Foundations of statistical natural language processing*. MIT press, 1st edition, 1999.

[38] Michael I Jordan and Tom M Mitchell. Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245):255–260, 2015.

[39] Aurélien Géron. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. " O'Reilly Media, Inc.", 2019.

[40] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[41] Nitesh V Chawla. Data mining for imbalanced datasets: An overview. *Data mining and knowledge discovery handbook*, pages 875–886, 2009.

[42] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[43] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[44] Naomi S Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3):175–185, 1992.

[45] David R Cox. The regression analysis of binary sequences. *Journal of the Royal Statistical Society: Series B (Methodological)*, 20(2):215–232, 1958.

[46] Irina Rish et al. An empirical study of the naive bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, volume 3, pages 41–46, 2001.

[47] Tin Kam Ho. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, volume 1, pages 278–282. IEEE, 1995.

[48] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[49] Robert E Schapire. Explaining adaboost. In *Empirical inference*, pages 37–52. Springer, 2013.

[50] Jerry Ye, Jyh-Herng Chow, Jiang Chen, and Zhaohui Zheng. Stochastic gradient boosted distributed decision trees. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pages 2061–2064, 2009.

[51] Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *The IEEE International Conference on Computer Vision*, December 2015.

[52] Wikimedia Foundation. Wikimedia downloads.

[53] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

[54] Karen Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 1972.

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

[56] Justyna Sarzynska-Wawer, Aleksander Wawer, Aleksandra Pawlak, Julia Szymanowska, Izabela Stefaniak, Michal Jarkiewicz, and Lukasz Okruszek. Detecting formal thought disorder by deep contextualized word representations. *Psychiatry Research*, 304:114135, 2021.

[57] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv:1908.10084*, 2019.

[58] Xingping Dong and Jianbing Shen. Triplet loss in siamese network for object tracking. In *Proceedings of the European conference on computer vision*, pages 459–474, 2018.

[59] Erik Kamsties, Daniel Berry, and Barbara Paech. Detecting ambiguities in requirements documents using inspections. In *Proceedings of the 1st Workshop on Inspection in Software Engineering*, 2001.

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

[61] Paola Spoletini, Alessio Ferrari, Muneera Bano, Didar Zowghi, and Stefania Gnesi. Interview review: An empirical study on detecting ambiguities in requirements elicitation interviews. In *Proceedings of the 24th Working Conference on Requirements Engineering: Foundation for Software Quality*, 2018.

[62] Siba Mishra and Arpit Sharma. On the use of word embeddings for identifying domain specific ambiguities in requirements. In *Proceedings of the 27th IEEE International Requirements Engineering Conference Workshops*, 2019.

[63] Daniel Toews and Leif Van Holland. Determining domain-specific differences of polysemous words using context information. In *Proceedings of the 25th Working Conference on Requirements Engineering: Foundation and Software Quality Workshops*, 2019.

[64] Vaibhav Jain, Ruchika Malhotra, Sanskar Jain, and Nishant Tanwar. Cross-domain ambiguity detection using linear transformation of word embedding spaces. In *Proceedings of the 26th Working Conference on Requirements Engineering: Foundation and Software Quality Workshops*, 2020.

[65] Chetan Arora, Mehrdad Sabetzadeh, Lionel Briand, and Frank Zimmer. Extracting domain models from natural-language requirements: approach and industrial evaluation. In *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems (MODELS'16)*, 2016.

[66] Amin Sleimi, Nicolas Sannier, Mehrdad Sabetzadeh, Lionel Briand, and John Dann. Automated extraction of semantic legal metadata using natural language processing. In *Proceedings of the 26th IEEE International Requirements Engineering Conference*, 2018.

[67] Barbara Strang. *Modern English Structure*. Edward Arnold, 2nd edition, 1968.

[68] Francis Chantree, Adam Kilgarriff, Anne De Roeck, and Alistair Willis. Disambiguating coordinations using word distribution information. In *Proceedings of the 5th International Conference on Recent Advances in Natural Language Processing*, 2005.

[69] Miriam Goldberg. An unsupervised model for statistically determining coordinate phrase attachment. In *Proceedings of the 37th annual meeting of the Association for Computational Linguistics*, 1999.

[70] P. Resnik. Semantic similarity in a taxonomy: An information-based measure and its application to problems of ambiguity in natural language. *Journal of Artificial Intelligence Research*, 11(1), 1999.

[71] Preslav Nakov and Marti Hearst. Using the web as an implicit training set: application to structural ambiguity resolution. In *Proceedings of the 5th conference on Human Language Technology and Empirical Methods in Natural Language Processing*, 2005.

[72] Anne De Roeck. Detecting dangerous coordination ambiguities using word distribution. In *Proceedings of the 6th International Conference on Recent Advances in Natural Language Processing*, 2007.

[73] Sri Tjong and Daniel Berry. Can rules of inferences resolve coordination ambiguity in natural language requirements specification? In *Proceedings of the 13th Workshop on Requirements Engineering*, 2008.

[74] Hui Yang, Alistair Willis, Anne De Roeck, and Bashar Nuseibeh. Automatic detection of nocuous coordination ambiguities in natural language requirements. In *Proceedings of the 10th IEEE/ACM international conference on Automated software engineering*, 2010.

[75] Sri Tjong and Daniel Berry. 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*, 2013.

[76] A. Kilgarriff. Thesauruses for natural language processing. In *Proceedings of the 1st International Conference on Natural Language Processing and Knowledge Engineering*, 2003.

[77] Hui Yang, Anne De Roeck, Alistair Willis, and Bashar Nuseibeh. A methodology for automatic identification of nocuous ambiguity. In *Proceedings of the 23rd International Conference on Computational Linguistics*, 2010.

[78] Akitoshi Okumura and Kazunori Muraki. Symmetric pattern matching analysis for English coordinate structures. In *Proceedings of the 4th Conference on Applied Natural Language Processing*, 1994.

[79] Eneko Agirre, Timothy Baldwin, and David Martínez. Improving parsing and PP attachment performance with sense information. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics*, 2008.

[80] Hiram Calvo and Alexander Gelbukh. Improving prepositional phrase attachment disambiguation using the web as corpus. In *Proceedings of the 8th Iberoamerican Congress on Progress in Pattern Recognition, Speech and Image Analysis*, 2003.

[81] Mitra Bokaei Hosseini, Rocky Slavin, Travis Breaux, Xiaoyin Wang, and Jianwei Niu. Disambiguating requirements through syntax-driven semantic analysis of information types. In *Proceedings of the 26th Working Conference on Requirements Engineering: Foundation for Software Quality*, 2020.

[82] Unnati Shah and Devesh Jinwala. Resolving ambiguities in natural language software requirements: A comprehensive survey. *SIGSOFT Software Engineering Notes*, 40(5), 2015.

[83] Cristina Ribeiro and Daniel Berry. 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.

[84] Fabrizio Fabbrini, Mario Fusani, Stefania Gnesi, and Giuseppe Lami. 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*, 2001.

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

[86] Benedikt Gleich, Oliver Creighton, and Leonid Kof. Ambiguity detection: Towards a tool explaining ambiguity sources. In *Proceedings of the 16th Working Conference on Requirements Engineering: Foundation for Software Quality*, 2010.

[87] Henning Femmer, Daniel Méndez Fernández, Stefan Wagner, and Sebastian Eder. Rapid quality assurance with requirements smells. *Journal of Systems and Software*, 123, 2017.

[88] Giuseppe Lami, Mario Fusani, and Gianluca Trentanni. QuARS: A pioneer tool for NL requirement analysis. In *From Software Engineering to Formal Methods and Tools, and Back*. Springer, 2019.

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

[90] Alistair Willis, Francis Chantree, and Anne De Roeck. Automatic identification of nocuous ambiguity. *Research on Language and Computation*, 6(3-4), 2008.

[91] Kenneth Church and Ramesh Patil. *Coping with Syntactic Ambiguity or How to Put the Block in the Box on the Table*. MIT Press, 1st edition, 1982.

[92] Patrick Pantel and Dekang Lin. An unsupervised approach to prepositional phrase attachment using contextually similar words. In *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*, 2000.

[93] Eneko Agirre, Oier de Lacalle, Christiane Fellbaum, Andrea Marchetti, Antonio Toral, and Piek Vossen. SemEval-2010 task 17: all-words word sense disambiguation on a specific domain. In *Proceedings of the 5th Workshop on Semantic Evaluations: Recent Achievements and Future Directions*, 2010.

[94] Michael Strube and Simone Ponzetto. WikiRelate! computing semantic relatedness using Wikipedia. In *Proceedings of the 21st national conference on Artificial intelligence*, 2006.

[95] Evgeniy Gabrilovich, Shaul Markovitch, et al. Computing semantic relatedness using wikipedia-based explicit semantic analysis. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, 2007.

[96] Angela Fogarolli. Word sense disambiguation based on Wikipedia link structure. In *Proceedings of the 3rd IEEE International Conference on Semantic Computing (ICSC'09)*, 2009.

[97] Spandana Gella, Carlo Strapparava, and Vivi Nastase. Mapping WordNet domains, WordNet topics and Wikipedia categories to generate multilingual domain specific resources. In *Proceedings of the 9th International Conference on Language Resources and Evaluation*, 2014.

[98] George Miller. WordNet: A lexical database for English. *Communications of the ACM*, 38(11), 1995.

[99] Christiane Fellbaum. *WordNet: An Electronic Lexical Database*. The MIT Press, 1st edition, 1998.

[100] Danqi Chen and Christopher Manning. A fast and accurate dependency parser using neural networks. In *Proceedings of the 18th Conference on Empirical Methods in Natural Language Processing*, 2014.

[101] Chetan Arora, Mehrdad Sabetzadeh, Lionel Briand, and Frank Zimmer. Automated extraction and clustering of requirements glossary terms. *IEEE Transactions on Software Engineering*, 43(10), 2017.

[102] David Newman, Jey Lau, Karl Grieser, and Timothy Baldwin. Automatic evaluation of topic coherence. In *Proceedings of the 8th annual conference of the North American chapter of the association for computational linguistics: Human language technologies*, 2010.

[103] Stefan Evert. Google web 1T 5-grams made easy (but not for the computer). In *Proceedings of the 8th annual conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies and the 6th Web as Corpus Workshop*, 2010.

[104] Chris Biemann, Felix Bildhauer, Stefan Evert, Dirk Goldhahn, Uwe Quasthoff, Roland Schäfer, Johannes Simon, Leonard Swiezinski, and Torsten Zesch. Scalable construction of high-quality web corpora. *Journal for Language Technology and Computational Linguistics*, 28(2), 2013.

[105] Tzu Yen, Jian Wu, Jim Chang, Joanne Boisson, and Jason Chang. WriteAhead: Mining grammar patterns in corpora for assisted writing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing, Proceedings of System Demonstrations*, 2015.

[106] Tobias Hawker. USYD: WSD and lexical substitution using the Web1T corpus. In *Proceedings of the 4th International Workshop on Semantic Evaluations*, 2007.

[107] D. Jurafsky and J. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall, 2nd edition, 2009.

[108] Georgiana Dinu and Mirella Lapata. Measuring distributional similarity in context. In *Proceedings of the 14th Conference on Empirical Methods in Natural Language Processing*, 2010.

[109] Laurel J Brinton. *The structure of modern English: A linguistic introduction*. John Benjamins Publishing, 2000.

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

[111] Richard Eckart de Castilho and Iryna Gurevych. A broad-coverage collection of portable NLP components for building shareable analysis pipelines. In *Proceedings of the Workshop on Open Infrastructures and Analysis Frameworks for HLT*, 2014.

[112] Torsten Zesch, Christof Müller, and Iryna Gurevych. Extracting lexical semantic knowledge from Wikipedia and Wiktionary. In *Proceedings of the 6th International Conference on Language Resources and Evaluation*, 2008.

[113] C Giuliano. jWeb1T: A library for searching the web 1T 5-gram corpus. Last accessed: August 2020.

[114] Muhua Zhu, Yue Zhang, Wenliang Chen, Min Zhang, and Jingbo Zhu. Fast and accurate shift-reduce constituent parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, 2013.

[115] Philip Resnik. Using information content to evaluate semantic similarity in a taxonomy. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, 1995.

[116] Hideki Shima. WS4J WordNet similarity for java. Last accessed: August 2020.

[117] J. Richard Landis and Gary G. Koch. An application of hierarchical kappa-type statistics in the assessment of majority agreement among multiple observers. *Biometrics*, 33(2), 1977.

[118] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(1), 2012.

[119] Geoffrey Leech. 100 million words of English. *English Today*, 9(1), 1993.

[120] J. Hirschberg and C.D. Manning. Advances in natural language processing. *Science*, 349(6245), 2015.

[121] Leo Breiman, Jerome Friedman, Richard Olshen, and Charles Stone. *Classification And Regression Trees*. Routledge, 1st edition, 1984.

[122] Yuan Tian and David Lo. A comparative study on the effectiveness of part-of-speech tagging techniques on bug reports. In *Proceedings of the 22nd IEEE International Conference on Software Analysis, Evolution, and Reengineering*, 2015.

[123] Jean Charbonnier and Christian Wartena. Using word embeddings for unsupervised acronym disambiguation. In *Proceedings of the 27th International Conference on Computational Linguistics*, 2018.

[124] Apache Software Foundation. Apache Maven. https://maven.apache.org/.

[125] Apache Software Foundation. Apache UIMA. https://uima.apache.org/.

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

[127] Edward Loper and Steven Bird. 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*, 2002.

[128] Matthew Honnibal, Ines Montani, Sofie Van Landeghem, and Adriane Boyd. spaCy: Industrial-strength Natural Language Processing in Python, 2020.

[129] M. McGill and G. Salton. *Introduction to Modern Information Retrieval*. McGraw-Hill, 1983.

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

[131] Dan Jurafsky. *Speech & language processing*. Pearson Education India, 2000.

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

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

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

[135] Chetan Arora, Mehrdad Sabetzadeh, Lionel Briand, Frank Zimmer, and Raul Gnaga. 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*, 2013.

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

[137] 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. Jupyter notebooks – a publishing format for reproducible computational workflows. In *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, 2016.

[138] Sallam Abualhaija, Davide Fucci, Fabiano Dalpiaz, Xavier Franch, and Alessio Ferrari. ReqEval: The shared task on anaphora ambiguity detection and disambiguation, 2020. last accessed: July 2021.

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

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

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

[142] Mohamed Osama, Aya Zaki-Ismail, Mohamed Abdelrazek, John Grundy, and Amani Ibrahim. Score-based automatic detection and resolution of syntactic ambiguity in natural language requirements. In *2020 IEEE International Conference on Software Maintenance and Evolution*, 2020.

[143] Yawen Wang, Lin Shi, Mingyang Li, Qing Wang, and Yun Yang. A deep context-wise method for coreference detection in natural language requirements. In *2020 IEEE 28th International Requirements Engineering Conference*, 2020.

[144] Joseph F McCarthy and Wendy G Lehnert. Using decision trees for coreference resolution. In *International Joint Conferences on Artificial Intelligence*, 1995.

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

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

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

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

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

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

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

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

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

[154] Yufang Hou. Bridging anaphora resolution as question answering. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020.

[155] Mitch Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 19(2), 1993.

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

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

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

[159] Saad Ezzini, Sallam Abualhaija, Chetan Arora, and Mehrdad Sabetzadeh. *"Online Annex (online)"*, 2021. Available at `https://tinyurl.com/yw29ff7r`, August 2021.

[160] Alessio Miaschi and Felice Dell'Orletta. 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, 2020.

[161] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing*, 2014.

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

[163] Xinyun Cheng, Xianglong Kong, Li Liao, and Bixin Li. A combined method for usage of nlp libraries towards analyzing software documents. In *International Conference on Advanced Information Systems Engineering*, 2020.

[164] Kevin Clark and Christopher D. Manning. Deep reinforcement learning for mention-ranking coreference models. In *Empirical Methods on Natural Language Processing*, 2016.

[165] Kevin Clark and Christopher D. Manning. Improving coreference resolution by learning entity-level distributed representations. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, 2016.

[166] Marcel Robeer, Garm Lucassen, Jan Martijn E.M. van der Werf, Fabiano Dalpiaz, and Sjaak Brinkkemper. Automated extraction of conceptual models from user stories via NLP. In *Proceedings of the 24th IEEE International Requirements Engineering Conference*, 2016.

[167] Chetan Arora, Mehrdad Sabetzadeh, Shiva Nejati, and Lionel Briand. An active learning approach for improving the accuracy of automated domain model extraction. *ACM Transactions on Software Engineering and Methodology*, 28(1), 2019.

[168] Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton, and Christopher D. Manning. 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*, 2020.

[169] Christopher Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven Bethard, and David McClosky. The Stanford CoreNLP natural language processing toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, 2014.

[170] 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. 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, 2020.

[171] 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. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019.

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

[173] Sallam Abualhaija, Davide Fucci, Fabiano Dalpiaz, and Xavier Franch. 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*, 2020.

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

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

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

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

[178] Felipe Quecole, Maisa Cristina Duarte, and Estevam Rafael Hruschka. Coupling for coreference resolution in a never-ending learning system. *Journal of Information and Data Management*, 9(2), 2018.

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

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

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

[182] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Neural Information Processing Systems Conference*, 2013.

[183] Daniel Berry. Evaluation of tools for hairy requirements and software engineering tasks. In *Proceedings of the 25th IEEE International Requirements Engineering Conference Workshops*, 2017.

[184] C. Arora, M. Sabetzadeh, A. Goknil, L. Briand, and F. Zimmer. Change impact analysis for natural language requirements: An NLP approach. In *23rd IEEE International Requirements Engineering Conference*, 2015.

[185] Fengbin Zhu, Wenqiang Lei, Chao Wang, Jianming Zheng, Soujanya Poria, and Tat-Seng Chua. Retrieving and reading: A comprehensive survey on open-domain question answering. *arXiv preprint arXiv:2101.00774*, 2021.

[186] Philipp Cimiano, Christina Unger, and John McCrae. Ontology-based interpretation of natural language. *Synthesis Lectures on Human Language Technologies*, 7(2):1–178, 2014.

[187] Yanchao Hao, Yuanzhe Zhang, Kang Liu, Shizhu He, Zhanyi Liu, Hua Wu, and Jun Zhao. An end-to-end model for question answering over knowledge base with cross-attention combining global knowledge. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 221–231, 2017.

[188] Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. Beir: A heterogenous benchmark for zero-shot evaluation of information retrieval models. *arXiv preprint arXiv:2104.08663*, 2021.

[189] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

[190] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

[191] Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. *Advances in Neural Information Processing Systems*, 33:5776–5788, 2020.

[192] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.

[193] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[194] Kevin Clark, Minh-Thang Luong, Quoc V Le, and Christopher D Manning. Electra: Pre-training text encoders as discriminators rather than generators. *arXiv preprint arXiv:2003.10555*, 2020.

[195] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv:1910.10683*, 2019.

[196] Ricardo Baeza-Yates, Berthier Ribeiro-Neto, et al. *Modern information retrieval*, volume 463. ACM press New York, 1999.

[197] Stephen Robertson and Hugo Zaragoza. *The probabilistic relevance framework: BM25 and beyond*. Now Publishers Inc, 2009.

[198] Christopher Manning, Prabhakar Raghavan, and Hinrich Schutze. *Introduction to Information Retrieval*. Cambridge University Press, 1st edition, 2008.

[199] Rodrigo Nogueira and Kyunghyun Cho. Passage re-ranking with bert. *arXiv preprint arXiv:1901.04085*, 2019.

[200] Yankai Lin, Haozhe Ji, Zhiyuan Liu, and Maosong Sun. Denoising distantly supervised open-domain question answering. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1736–1745, 2018.

[201] Rajarshi Das, Shehzaad Dhuliawala, Manzil Zaheer, and Andrew McCallum. Multi-step retriever-reader interaction for scalable open-domain question answering. *arXiv preprint arXiv:1905.05733*, 2019.

[202] Omar Khattab, Christopher Potts, and Matei Zaharia. Relevance-guided supervision for openqa with colbert. *Transactions of the Association for Computational Linguistics*, 9:929–944, 2021.

[203] Zhuosheng Zhang, Hai Zhao, and Rui Wang. Machine reading comprehension: The role of contextualized language models and beyond. *arXiv preprint arXiv:2005.06249*, 2020.

[204] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*, 2019.

[205] Shanshan Liu, Xin Zhang, Sheng Zhang, Hui Wang, and Weiming Zhang. Neural machine reading comprehension: Methods and trends. *Applied Sciences*, 9(18):3698, 2019.

[206] Rakesh Chada and Pradeep Natarajan. Fewshotqa: A simple framework for few-shot learning of question answering tasks using pre-trained text-to-text models. *arXiv preprint arXiv:2109.01951*, 2021.

[207] Manda Sai Divya and Shiv Kumar Goyal. Elasticsearch: An advanced and quick search technique to handle voluminous data. *Compusoft*, 2(6):171, 2013.

[208] Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. BEIR: A heterogeneous benchmark for zero-shot evaluation of information retrieval models. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.

[209] Brown Dorian, Jain Sarthak, Novotný Vít, and nlp4whp. dorianbrown/rank_bm25:, February 2022.

[210] Malte Pietsch, Tanay Soni, Branden Chan, Timo Möller, and Bogdan Kostić. haystack.deepset.ai:.

[211] Sallam Abualhaija, Chetan Arora, Mehrdad Sabetzadeh, Lionel Briand, and Eduardo Vaz. A machine learning-based approach for demarcating requirements in textual specifications. In *Proceedings of the 27th IEEE International Requirements Engineering Conference*, 2019.

[212] Washington Cunha, Vítor Mangaravite, Christian Gomes, Sérgio Canuto, Elaine Resende, Cecilia Nascimento, Felipe Viegas, Celso França, Wellington Santos Martins, Jussara M Almeida, et al. On the cost-effectiveness of neural and non-neural approaches and representations for text classification: A comprehensive comparative study. *Information Processing & Management*, 58(3):102481, 2021.

[213] Sugandha Malviya, Michael Vierhauser, Jane Cleland-Huang, and Smita Ghaisas. What questions do requirements engineers ask? In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 100–109. IEEE, 2017.

[214] Mujahid Sultan and Andriy Miranskyy. Ordering interrogative questions for effective requirements engineering: The w6h pattern. In *2015 IEEE Fifth International Workshop on Requirements Patterns*, pages 1–8. IEEE, 2015.

[215] Sara Pérez-Soler, Esther Guerra, and Juan de Lara. Creating and migrating chatbots with conga. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings*, pages 37–40. IEEE, 2021.

[216] Neng Zhang, Qiao Huang, Xin Xia, Ying Zou, David Lo, and Zhenchang Xing. Chatbot4qr: Interactive query refinement for technical question retrieval. *IEEE Transactions on Software Engineering*, 2020.

[217] Aakash Bansal, Zachary Eberhart, Lingfei Wu, and Collin McMillan. A neural question answering system for basic questions about subroutines. In *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering*, pages 60–71. IEEE, 2021.

[218] Tianyong Hao, Xinxin Li, Yulan He, Fu Lee Wang, and Yingying Qu. Recent progress in leveraging deep learning methods for question answering. *Neural Computing and Applications*, pages 1–19, 2022.

[219] Abdulganiyu Abdu Yusuf, Feng Chong, and Mao Xianling. An analysis of graph convolutional networks and recent datasets for visual question answering. *Artificial Intelligence Review*, pages 1–24, 2022.

[220] Hai Jin, Yi Luo, Chenjing Gao, Xunzhu Tang, and Pingpeng Yuan. Comqa: Question answering over knowledge base via semantic matching. *IEEE Access*, 7:75235–75246, 2019.

[221] Dennis Diefenbach, Andreas Both, Kamal Singh, and Pierre Maret. Towards a question answering system over the semantic web. *Semantic Web*, 11(3):421–439, 2020.

[222] Bolanle Ojokoh and Emmanuel Adebisi. A review of question answering systems. *Journal of Web Engineering*, 17(8):717–758, 2018.

[223] Li Jing, Caglar Gulcehre, John Peurifoy, Yichen Shen, Max Tegmark, Marin Soljacic, and Yoshua Bengio. Gated orthogonal recurrent units: On learning to forget. *Neural computation*, 31(4):765–783, 2019.

[224] Aziguli Wulamu, Zhenqi Sun, Yonghong Xie, Cong Xu, and Alan Yang. An improved end-to-end memory network for qa tasks. *CMC-COMPUTERS MATERIALS & CONTINUA*, 60(3):1283–1295, 2019.

[225] Qiyu Ren, Xiang Cheng, and Sen Su. Multi-task learning with generative adversarial training for multi-passage machine reading comprehension. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 8705–8712, 2020.

[226] Tetiana Parshakova, Francois Rameau, Andriy Serdega, In So Kweon, and Dae-Shik Kim. Latent question interpretation through variational adaptation. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 27(11):1713–1724, 2019.

[227] Eduardo Cortes, Vinicius Woloszyn, Arne Binder, Tilo Himmelsbach, Dante Barone, and Sebastian Möller. An empirical comparison of question classification methods for question answering systems. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pages 5408–5416, 2020.

[228] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

[229] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

[230] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. Natural questions: a benchmark for

question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466, 2019.

[231] Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. Ms marco: A human generated machine reading comprehension dataset. In *CoCo@ NIPS*, 2016.

[232] Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. *arXiv preprint arXiv:1705.03551*, 2017.

[233] Heyan Huang, Xiaochi Wei, Liqiang Nie, Xianling Mao, and Xin-Shun Xu. From question to text: Question-oriented feature attention for answer selection. *ACM Transactions on Information Systems (TOIS)*, 37(1):1–33, 2018.

[234] Chia-Hsuan Lee, Hung-yi Lee, Szu-Lin Wu, Chi-Liang Liu, Wei Fang, Juei-Yang Hsu, and Bo-Hsiang Tseng. Machine comprehension of spoken content: Toefl listening test and spoken squad. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 27(9):1469–1480, 2019.

[235] Yuyu Zhang, Hanjun Dai, Zornitsa Kozareva, Alexander J Smola, and Le Song. Variational reasoning for question answering with knowledge graph. In *Thirty-second AAAI conference on artificial intelligence*, 2018.

[236] Liwen Zhang, John Winn, and Ryota Tomioka. Gaussian attention model and its application to knowledge base embedding and question answering. *arXiv preprint arXiv:1611.02266*, 2016.

[237] Junqing He, Mingming Fu, and Manshu Tu. Applying deep matching networks to chinese medical question answering: a study and a dataset. *BMC medical informatics and decision making*, 19(2):91–100, 2019.

[238] Liangming Pan, Wenqiang Lei, Tat-Seng Chua, and Min-Yen Kan. Recent advances in neural question generation. *arXiv preprint arXiv:1905.08949*, 2019.

[239] Vishwajeet Kumar, Yuncheng Hua, Ganesh Ramakrishnan, Guilin Qi, Lianli Gao, and Yuan-Fang Li. Difficulty-controllable multi-hop question generation from knowledge graphs. In *International Semantic Web Conference*, pages 382–398. Springer, 2019.

[240] Nelson F Liu, Tony Lee, Robin Jia, and Percy Liang. Can small and synthetic benchmarks drive modeling innovation? a retrospective study of question answering modeling approaches. *arXiv preprint arXiv:2102.01065*, 2021.

[241] Max Bartolo, Tristan Thrush, Robin Jia, Sebastian Riedel, Pontus Stenetorp, and Douwe Kiela. Improving question answering model robustness with synthetic adversarial data generation. *arXiv preprint arXiv:2104.08678*, 2021.

[242] Adam D Lelkes, Vinh Q Tran, and Cong Yu. Quiz-style question generation for news stories. In *Proceedings of the Web Conference 2021*, pages 2501–2511, 2021.

[243] Shrey Gupta, Anmol Agarwal, Manas Gaur, Kaushik Roy, Vignesh Narayanan, Ponnurangam Kumaraguru, and Amit Sheth. Learning to automate follow-up question generation using process knowledge for depression triage on reddit posts. *arXiv preprint arXiv:2205.13884*, 2022.