CAPITAL UNIVERSITY OF SCIENCE AND TECHNOLOGY, ISLAMABAD

Automated Detection of Ambiguities in Natural Language Software Requirements

by

Muhammad Arsalan Iltaf

A thesis submitted in partial fulfillment for the degree of Master of Science

in the

Faculty of Computing
Department of Computer Science

2025

CERTIFICATE OF APPROVAL

Automated Detection of Ambiguities in Natural Language Software Requirements

by

Muhammad Arsalan Iltaf

(MCS231011)

THESIS EXAMINING COMMITTEE

S. No.	Examiner	Name	Organization
(a)	External Examiner	Dr. Rabeeh Ayaz Abbasi	QAU, Islamabad
(b)	Internal Examiner	Dr. Nadeem Anjum	CUST, Islamabad

Dr. Aamer Nadeem
Thesis Supervisor
December, 2025

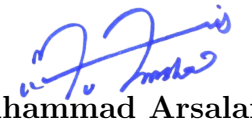
Dr. Muhammad Masroor Ahmed
Head
Dept. of Computer Science
December, 2025

Dr. Muhammad Abdul Qadir
Dean
Faculty of Computing
December, 2025

Author's Declaration

I, **Muhammad Arsalan Iltaf** hereby state that my MS thesis titled "**Automated Detection of Ambiguities in Natural Language Software Requirements**" is my own work and has not been submitted previously by me for taking any degree from Capital University of Science and Technology, Islamabad or anywhere else in the country/abroad.

At any time if my statement is found to be incorrect even after my graduation, the University has the right to withdraw my MS Degree.



(Muhammad Arsalan Iltaf)

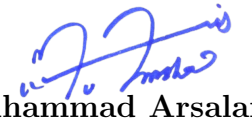
Registration No: MCS231011

Plagiarism Undertaking

I solemnly declare that research work presented in this thesis titled "**Automated Detection of Ambiguities in Natural Language Software Requirements**" is solely my research work with no significant contribution from any other person. Small contribution/help wherever taken has been duly acknowledged and that complete thesis has been written by me.

I understand the zero tolerance policy of the HEC and Capital University of Science and Technology towards plagiarism. Therefore, I as an author of the above titled thesis declare that no portion of my thesis has been plagiarized and any material used as reference is properly referred/cited.

I undertake that if I am found guilty of any formal plagiarism in the above titled thesis even after award of MS Degree, the University reserves the right to withdraw/revoke my MS degree and that HEC and the University have the right to publish my name on the HEC/University website on which names of students are placed who submitted plagiarized work.



(Muhammad Arsalan Iltaf)

Registration No: MCS231011

Acknowledgement

I would like to express my gratitude to Allah Almighty bestowing me with the wisdom and strength in order to accomplish the writing of this dissertation. Pursuing a master's degree at Capital University of Science and Technology has been an extraordinary yet demanding journey, and I appreciate the growth and experiences it has brought me.

I would like to express my sincere gratitude to my diligent supervisor Dr. Aamer Nadeem who has been a great help in guiding, supporting and imparting considerable knowledge that helped so much in the formulation of this research. I really appreciate his constant support or rather encouragement, his inspiration, and not to mention his patience. His pieces of advice and useful comments have contributed to the success of this thesis considerably.

I would also wish to appreciate my family and more so my parents, who have provided unending support, which has been one of the greatest catalysts in realization of this milestone. I thank my friends and classmates who assisted me by helping share knowledge and resources to be used in conduction of this research.

Lastly, I have to thank my wife to the best that I can. I have been a strong individual, thanks to her empowerment, encouragement and comprehension. She has been my source of light as a result of her confidence in my potentials and I am very grateful to be surrounded with her on this academic journey.

(Muhammad Arsalan Iltaf)

Abstract

Natural language remains the predominant medium for expressing software requirements, yet it is inherently susceptible to quality issues that affect precision and clarity. Among these issues, anaphoric ambiguity, referential ambiguity, unverifiability, and subjectivity appear frequently across real-world specifications and require systematic handling to support early requirements quality assessment. This thesis develops four dedicated detection approaches one for each smell type based on a combination of natural language processing techniques and machine learning models.

The methodology integrates contextual embeddings, weighted TF–IDF representations, and targeted linguistic indicators such as pronoun usage, modal verbs, vague expressions, and structural cues. Anaphoric ambiguity is detected using a hybrid pipeline that fuses DistilBERT embeddings with linguistic features and an optimized XGBoost classifier. Referential ambiguity and unverifiability are identified through weighted TF–IDF and handcrafted linguistic features combined with Random Forest models and calibrated decision thresholds. Subjectivity is addressed using an enhanced rule-based procedure with an optional machine learning classifier. Experiments were conducted on three publicly available datasets: DAMIR-PURE, GitHub (ReqEval), and Zenodo.

The results show strong performance across all four smells, with weighted F1-scores typically ranging from 0.88 to 0.95 and additional gains achieved through probability threshold optimization. These findings demonstrate that combining contextual semantic cues with lightweight linguistic features provides an effective and computationally efficient basis for detecting key requirement smells, contributing toward more reliable and interpretable requirements analysis practices.

Contents

Author's Declaration iii

Plagiarism Undertaking iv

Acknowledgement v

Abstract vi

List of Figures x

List of Tables xi

Abbreviations xii

1 Introduction 1

1.1 Requirement Specification Methods . . . 2
1.1.1 Natural Language . . . 2
1.1.2 Formal Methods . . . 2
1.1.3 Semi Formal Techniques . . . 2
1.1.4 Visual Modeling Techniques . . . 3
1.2 Advantages of Natural Language Specifications . . . 3
1.2.1 Availability . . . 3
1.2.2 Expressiveness . . . 4
1.2.3 Collaboration . . . 4
1.2.4 Flexibility . . . 4
1.2.5 Cost Effectiveness . . . 4
1.3 Quality Issues in Natural Language Specifications . . . 4
1.3.1 Ambiguity . . . 5
1.3.2 Clarity and Precision Issues . . . 7
1.3.3 Structural and Linguistic Issues . . . 9
1.3.4 Content and Consistency Issues . . . 9
1.4 Existing Solutions to Detect Smells . . . 11
1.4.1 Manual Reviews and Inspection . . . 11
1.4.2 Rule Based Methods . . . 11
1.4.3 Machine Learning and Deep Learning Methods . . . 12

1.4.4 Hybrid Solutions . . . 12
1.5 Problem Statement . . . 13
1.6 Research Objectives . . . 14
1.7 Research Questions . . . 14
1.8 Methodology . . . 15
1.9 Thesis Structure . . . 16

2 Literature Review **18**
2.1 Scope and Objectives . . . 18
2.2 Classification of Smells in Natural Language Requirements . . . 20
2.2.1 Ambiguity . . . 20
2.2.2 Subjectivity . . . 24
2.2.3 Verifiability . . . 25
2.3 Smells Detection Methods . . . 25
2.3.1 Manual Reviews . . . 26
2.3.2 Rule Based Methods . . . 28
2.3.3 Machine Learning and Deep Learning Methods . . . 30
2.3.4 Hybrid Methods . . . 33
2.4 Gap Analysis . . . 35

3 Proposed Methodology **36**
3.1 Proposed Methodology for Anaphoric Ambiguity Detection . . . 37
3.1.1 Data Preparation and Preprocessing . . . 38
3.1.2 Linguistic Feature Engineering . . . 38
3.1.3 Contextual Semantic Embedding Extraction . . . 40
3.1.4 Hybrid Feature Fusion and Pipeline Construction . . . 40
3.1.5 Classification Using XGBoost . . . 40
3.1.6 Model Calibration and Threshold Optimization . . . 41
3.1.7 Output Generation and Implementation Environment . . . 41
3.2 Proposed Methodology for Referential Ambiguity Detection . . . 42
3.2.1 Data Preparation and Preprocessing . . . 42
3.2.2 Advanced Feature Engineering . . . 44
3.2.3 Feature Representation and Pipeline Construction . . . 45
3.2.4 Classification with Random Forest . . . 48
3.2.5 Model Training and Evaluation . . . 49
3.2.6 Probability Calibration and Threshold Optimization . . . 49
3.3 Proposed Methodology for Unverifiability Detection . . . 50
3.3.1 Data Preparation and Preprocessing . . . 52
3.3.2 Advanced Feature Engineering . . . 52
3.3.3 Feature Representation and Pipeline Construction . . . 54
3.3.4 Pipeline Implementation and Feature Fusion . . . 56
3.3.5 Classifier Configuration and Training Procedure . . . 57
3.3.6 Probability Calibration and Threshold Optimization . . . 58

3.3.7 Classification with Random Forest . . . 59
3.3.8 Model Training and Evaluation . . . 59
3.4 Proposed Methodology for Subjectivity Detection . . . 60
3.4.1 Data Preparation and Preprocessing . . . 60
3.4.2 Feature Engineering . . . 62
3.4.3 Rule Based Detection . . . 63
3.4.4 Machine Learning Classifier . . . 66
3.4.5 Model Training and Evaluation . . . 67
3.4.6 Hybrid Subjectivity Detection Process . . . 68
3.5 Saving, Reporting and Visualization . . . 69
3.6 Evaluation Metrics . . . 70

4 Experiments and Results **73**
4.1 Introduction . . . 74
4.2 Experimental Design . . . 74
4.2.1 Dataset Preparation . . . 74
4.2.2 Experiment Protocol . . . 76
4.3 Anaphoric Ambiguity Detection Results . . . 77
4.4 Referential Ambiguity Detection Results . . . 78
4.5 Unverifiability Detection Results . . . 80
4.6 Subjectivity Detection Results . . . 81
4.7 Comparative Analysis . . . 83

5 Conclusion and Future Work **85**
5.1 Strengths of the Proposed Approach . . . 86
5.2 Limitations of the Current Work . . . 88
5.3 Future Enhancements . . . 89

Bibliography **91**

List of Figures

1.1 Thesis Structure 17

2.1 Classification of Requirement Smells 21
2.2 Example: Anaphoric Ambiguity [8] 23

3.1 Anaphoric Ambiguity Detection Flow Diagram 39
3.2 Referential Ambiguity Detection Flow Diagram 43
3.3 Unverifiability Detection Flow Diagram 51
3.4 Subjectivity Detection Flow Diagram 61

4.1 Summary of Datasets Used for Requirement Smells Detection . . . 76
4.2 Anaphoric Ambiguity - Results 78
4.3 Referential Ambiguity - Results 79
4.4 Unverifiability - Results 81
4.5 Results for Subjectivity Detection 82
4.6 Comparative Analysis 84

List of Tables

2.1 Manual Reviews 27
2.2 Rule Based Methods 29
2.3 Machine Learning/Deep Learning Methods 32
2.4 Hybrid Methods 34

3.1 Performance of Different Text and Numeric Feature Weights for Referential Ambiguity Detection 47

4.1 Summary of Datasets Used for Requirements Smell Detection . . . 75
4.2 Anaphoric Ambiguity Detection Results 78
4.3 Referential Ambiguity Detection Results 79
4.4 Unverifiability Detection Results 80
4.5 Evaluation Results for Subjectivity Detection 82
4.6 Comparative Analysis 84

Abbreviations

AI	Artificial Intelligence
ARM	Automated Requirements Measurement
BERT	Bidirectional Encoder Representations from Transformers
CISQ	Consortium for Information & Software Quality
CNL	Controlled Natural Language
DAMIR-PURE	Dataset for Anaphoric Ambiguity In Requirements - Public Requirements
DL	Deep Learning
ML	Machine Learning
NL	Natural Language
NLP	Natural Language Processing
NL-SRS	Natural Language Software Requirements Specifications
QuARS	Quality Analyzer for Requirement Specifications
RE	Requirements Engineering
RETA	REquirements Template Analyzer
SMOTE	Synthetic Minority Oversampling Technique
SRS	Software Requirements Specification
SVM	Support Vector Machine
TF-IDF	Term Frequency-Inverse Document Frequency
UML	Unified Modeling Language
XGBoost	eXtreme Gradient Boosting

Chapter 1

Introduction

Requirements engineering (RE) is a critical discipline in software engineering, focusing on the elicitation, analysis, specification, validation, and management of requirements to ensure that software systems satisfy stakeholder needs and operational demands. As the first stage in the software development lifecycle, RE underpins design, implementation, testing, and maintenance [1]. Effective requirements specifications are inherently correlated with project success, while poor specifications are among the leading causes of project failures, rework, and cost overruns [2, 3].

Recent industry reports highlight the economic impact of inadequate requirements. For instance, the Consortium for Information and Software Quality (CISQ) reported that poor software quality including requirements-related problems cost the U.S. economy $2.41 trillion in 2022, of which $1.52 trillion was due to technical debt [2]. More than half of software project issues have been linked to requirements-related defects, underscoring the critical role of RE [2].

Given these stakes, it becomes paramount to ensure that requirements are properly documented and effectively communicated. A requirements specification is a formal document (or collection of documents) that clearly defines the functions, features, constraints, and qualities a software system must provide [4]. Serving

as a contract between stakeholders and developers, it establishes a shared understanding of system objectives and operational conditions. Ideally, specifications should be complete, unambiguous, consistent, modifiable, verifiable, and traceable [5, 6]. They serve as a key reference point throughout the software lifecycle, guiding design decisions, implementation, verification, and maintenance [6].

1.1 Requirement Specification Methods

There are also different methods to document requirements, and they have individual strong and weak sides:

1.1.1 Natural Language

This is the most widely used approach and NL specifications can be read and understood, as well as, used without any special training. They favor recording the various types of requirements which include functional, non-functional and domain-specific requirements [1, 4, 6].

1.1.2 Formal Methods

These are based upon mathematical rigorous and precise notations, e.g. Z, B, or Petri nets, in order to remove ambiguity and make verification of a formal nature. Precise formal techniques cannot be applied without much expertise thus making it impossible to use them in non-technical projects [7].

1.1.3 Semi Formal Techniques

Processes such as Unified Modeling Language (UML), use case diagrams, and data flow diagram interact with organization in a mixture of a format that can be read and precise at the expense of flexibility. Their use is common however they may

not be fully applicable in the case of ambiguity when dealing with complicated systems [6].

1.1.4 Visual Modeling Techniques

Flowcharts, state charts and entity relationship diagrams are visual modeling techniques which are easier to understand systems behavior. Although they work well when covering high-level overview, they are not as detailed as necessary in coming up with comprehensive specifications especially when it comes to non-functional requirements [6].

The use of natural language prevails because of flexibility and stakeholder-friendly factors, particularly in the agile approach and in projects involving a diverse team. Nonetheless, it is prone to the phenomenon of "smells" - deficiencies compromising clarity and their verifiability [3, 8]. The prevention and detection methods should therefore be strong [3, 8].

1.2 Advantages of Natural Language Specifications

There are a number of strong reasons why natural language specifications have become common in the requirements engineering:

1.2.1 Availability

NL does not include any specialized training and allows the stakeholders involved in the requirements process such as clients, end-users, domain experts and developers to engage in the requirements process without having to learn complex notations [4].

1.2.2 Expressiveness

NL enhances expression of a broad spectrum of requirements, functional behaviors to non-functional attribute as well as domain specific constraints. This is necessary in bridging the abstract concepts or complex concepts [9].

1.2.3 Collaboration

NL also promotes collaboration with interdisciplinary teams because they have a shared medium of communication. When working on agile development, natural language style user stories can be used to do iterative and stakeholder requirements definition [9].

1.2.4 Flexibility

NL can be applied in different project settings, in small-scale usage as well as large-scale, domain-specific systems, with no need to involve special tools or approaches [10].

1.2.5 Cost Effectiveness

NL specifications can be made and viewed using common tools. This increases the accessibility of people to use them as opposed to their formal counterparts which require training and specific software [10].

Besides these benefits, the tendency of majority of smells in NL requirements, as it is observed in the CISQ reports, results in substantial economic losses, which explains the necessity of automated quality assurance [2, 3].

1.3 Quality Issues in Natural Language Specifications

Natural language (NL) specifications are prone to several quality problems, commonly referred to as *smells*, which can compromise the clarity, consistency, and verifiability of requirements. These smells often lead to miscommunication, misinterpretation, and costly errors, contributing to an estimated loss of $2.41 trillion in 2022 [2]. A comprehensive Requirements Smell Taxonomy categorizes these

defects into four main types: Ambiguity, Clarity and Precision Issues, Structural and Linguistic Issues, and Content and Consistency Issues [3]. This taxonomy provides a structured framework for identifying defects in software requirements that, if unaddressed, may result in inefficiencies, cost overruns, project failure, or misinterpretation [3]. Ambiguity may arise from multiple linguistic interpretations. Clarity and precision issues occur when requirements contain subjective or untestable statements. Structural and linguistic issues result from grammatical errors or poor organization, while content and consistency issues stem from logical inconsistencies or irrelevant information. Early identification and correction of these problems enhance testability, reduce rework, and improve overall quality, particularly when supported by text-based alerts such as NLP-based identifiers [11].

1.3.1 Ambiguity

Ambiguity is one of the basic issues natural language requirements have to cope with: they appear in instances where a statement is readable in more than one sense. Unclear requirements make the scenario confusing to the stakeholders as well as the developers and, thus, cause the lack of uniformity, errors, and wastage of time and money due to reworkings [1, 12]. The natural language needs tend to be affected by various types of ambiguity, and these are divided into four categories: lexical, syntactic, semantic, and pragmatic [13, 14]. Lexical Ambiguity occurs when a single word can have more than one meaning depending on the context in which it is used. This ambiguity is particularly problematic in requirements, where precise interpretation is essential. Lexical ambiguity is further divided into two subtypes: homonymy and polysemy [13, 15]. Homonymy Ambiguity arises when a word has entirely different meanings that are unrelated. For example: “The system retrieves data from the bank.” Here, the term bank could refer to either a financial institution or a river bank, creating confusion about the system’s domain of operation [13]. Polysemy Ambiguity occurs when a word has multiple related meanings. For example: “The user connects the device to the port.” In this case, port could mean either a USB port or a network port.

Both interpretations are valid and context-dependent [15]. Syntactic Ambiguity (also called structural ambiguity) arises when a sentence's structure allows multiple grammatical interpretations. The ambiguity is not in the individual words but in how they are combined. This type of ambiguity is common in long or complex requirement statements [14, 16]. Two major subtypes are attachment ambiguity and coordination ambiguity.

Attachment Ambiguity occurs when it is unclear which part of the sentence a phrase modifies. For example: "The system will store the data on the server that has high capacity." Here, that has high capacity may refer either to the system or to the server [14].

Coordination Ambiguity occurs when conjunctions such as and or or can group multiple phrases in different ways. For example: "The system will enable loading files and deleting the files and logs." It is ambiguous whether logs are deleted together with files, or separately [13].

Semantic ambiguity arises when a sentence's meaning is unclear due to uncertainty about the semantic relationships between words or phrases. Unlike lexical ambiguity, which concerns individual words, semantic ambiguity affects the overall interpretation of meaning in context. In software requirements, it can cause different stakeholders to derive inconsistent understandings of system behavior [15, 17]. Two common subtypes of semantic ambiguity are scope ambiguity and anaphoric ambiguity.

Scope Ambiguity occurs when the range of a quantifier, negation, or modal verb is unclear. For example: "All users cannot access the admin panel." This can mean either (1) no user can access it, or (2) not all users can access it (some can) [18, 19].

Anaphoric Ambiguity arises when pronouns or referring expressions (e.g., it, this, they) can refer to multiple antecedents. For example: "The system notifies the user about the error and stores it in the log." Here, it may refer either to the error or to the notification [8, 20].

Pragmatic Ambiguity arises when the intended meaning of a requirement depends on contextual or situational interpretation rather than explicit linguistic structure. In such cases, understanding the statement requires external knowledge or assumptions not stated in the text. As a result, different readers or stakeholders may interpret the same requirement differently [13, 15].

A prominent subtype of pragmatic ambiguity is referential ambiguity, which occurs when the reference of a pronoun, noun phrase, or entity is unclear. i.e., when it is ambiguous which object or subject is being referred to. Such ambiguity is frequent in natural language requirements, particularly when multiple potential antecedents exist within the same sentence [8, 11, 20]. Referential Ambiguity: Example 1: "The system records the transaction and sends it to the user." In this sentence, the pronoun it could refer either to the transaction or to the record, making the intended meaning ambiguous [20]. Example 2: "The system sends the data to the module and stores it in the database." Here, it may refer either to the data or to the module, leading to multiple valid interpretations of the same requirement [8].

1.3.2 Clarity and Precision Issues

Clarity and Precision Issues emphasise subjectivity, unverifiability, vagueness and complexity.

Subjectivity employs qualitative descriptions (my words - user-friendly, easy to use) which have no objective acceptance criteria; these should be substituted by measureable constraints (my words-no more than three navigation actions) [21, 22]. For example "The system should provide a user-friendly interface." The adjective user-friendly is subjective and open to personal interpretation. What one stakeholder perceives as user-friendly (e.g., visual design) may differ from another's expectation (e.g., navigation simplicity). This type of linguistic quality issue is categorized as a subjectivity smell, because it expresses an opinion-based judgment rather than an objective quality [3, 23].

Unverifiability efers to requirements expressed as universal or unconditional statements that cannot be objectively tested. Such requirements often contain terms that lack measurable criteria or quantifiable thresholds, preventing consistent evaluation during validation or testing. For example "The system shall respond quickly to user requests." The adverb quickly lacks a defined time limit, making the requirement impossible to verify in practice. A verifiable version would specify a measurable target, for example, "The system shall respond within 2 seconds." Unverifiable requirements hinder objective assessment of compliance and were identified as a critical quality problem in requirements engineering [6, 11, 12, 22].

Vagueness occurs when a requirement includes words or phrases that are open to interpretation or lack precise meaning. Such statements make it difficult for stakeholders to develop a shared understanding of what the system must achieve. "The system should use appropriate encryption methods to protect data." The term appropriate is unclear; it does not specify which encryption standards or algorithms (e.g., AES-256, RSA). This lack of precision can lead to inconsistent understanding and implementation. This issue was identified as a vagueness-related quality problem in software requirements [1, 3].

Complexity refers to requirements that are mentally demanding or difficult to interpret because they contain several conditions, operations, or system behaviors expressed together. When a single requirement attempts to describe multiple functions or decisions, it becomes harder to analyze, implement, and maintain effectively. Excessive complexity increases the likelihood of misunderstanding and errors during development and testing [11, 21]. For example, "If the user is logged in and the payment is successful and the account balance is greater than zero, then the system shall update the order status and send a confirmation email." This statement combines multiple logical conditions and outcomes in a single requirement, making it cognitively difficult to understand and trace during implementation [11]. "The system shall continuously monitor performance, log events, notify the administrator in case of anomalies, and attempt automatic recovery." This sentence describes multiple system behaviors that should ideally be separated into distinct requirements to reduce complexity and improve clarity [12, 21].

1.3.3 Structural and Linguistic Issues

Structural and Linguistic Issues emphasis punctuation and modality.

Punctuation errors or omissions change sentence meaning or introduce run-ons that obscure intent (for example, missing commas or poorly placed parentheses can make one sentence describe multiple unrelated behaviors) [12]. Example 1: “The system shall not delete, the user data.” [24]. Example 2: “The system records inputs, the data are verified, and stored.” [23]. Example 3: “After validation the data are stored.” [24]. Incorrect punctuation can also lead to syntactic parsing errors or change the semantic dependency between clauses [23, 24].

Modality just means the usage of modal verbs (e.g. ”shall”, ”may”) and the normative effect thereof: undefined modality does not distinguish whether a constraint is mandatory/recommendation/optional to implementers, modal verbs explicitly distinguish the normative character of mandatory constraints and the use of explicitly explaining optional features [25, 26]. Example 1: “The system should log user activities.” [12]. Example 2: “The system may delete temporary files.” [27]. Example 3: “The module shall process data; it should also verify the checksum.” [3]. Consistent use of shall for mandatory and testable requirements is essential to avoid ambiguity and ensure clarity [1, 27].

This taxonomy distinguishes the structural-focused problems as they are specifically classified under the punctuation and modality so as to clearly distinguish between them and the more general linguistic issues like passive voice, repetition, or long sentences etc. [3].

1.3.4 Content and Consistency Issues

The problems of content and consistency of software requirements may present serious difficulties to the development process [3, 13].

Inconsistency may be anonymous where two or more requirements are in conflict (like one rule sets the maximum password length of 10 characters whereas another

sets the password limit length at 8 characters) [21, 28]. Example 1: R1: "The system shall lock the user account after three failed login attempts." R2: "The system shall allow users unlimited login attempts." [28]. Example 2: "The application shall store all user data locally." "User data shall be stored on the cloud." [29]. These contradictions make it unclear how the system should function and are considered major content-related defects.

Redundancy is repetitive or overlapping requirements in different words in the document, which cause maintenance problems [3, 30]. Example 1: R1: "The system shall generate a monthly report." R2: "The system shall produce a report every month." [11]. Example 2: "The software shall notify the user of any errors." "An error message shall be displayed to inform the user of any issues." [3]. Although both pairs of requirements convey the same meaning, redundancy can complicate requirement traceability and version control, as changes made to one statement might not be reflected in its duplicate, potentially leading to misalignment in documentation or implementation [27, 31]

Incompleteness refers to the omission of required information (i.e. who, when, format, units, frequency) e.g. report on the generation of reports without a specification of which reports or their frequency [21, 32, 33]. for example "The system shall send a notification if the user's account balance becomes less than zero." Example 1: "The system shall generate a report." [34]. This statement is incomplete because it does not specify what type of report, for whom, or under what conditions. Example 2: "If the data are invalid, an error message is displayed." [12]. Here, the responsible actor or error-handling behavior is not defined, making the requirement unclear.

Obsolescence is requirements that refer to obsolete platforms or versions (e.g., the system shall support Internet Explorer 8), which must be replaced with a generalized requirement (e.g., the system shall support current versions of Internet Explorer) or otherwise placed in maintenance-friendly words (e.g., the system shall support other current versions of Internet Explorer) [11]. Example 1: "The system shall support Internet Explorer 8." [35]. This requirement is obsolete because the

referenced browser is outdated and no longer supported. Example 2: "The mobile app shall use the legacy authentication API." [23]. This statement refers to a deprecated API, which introduces technical debt and future maintenance issues.

1.4 Existing Solutions to Detect Smells

There exist a number of strategies devised to decrease the reliance of smells within the contexts of natural language specifications in attempts to increase the degree of clarity, coherency, and verification. There are manual methods, rule-based, machine learning and hybrid methods of these solutions, but all of them are limited somehow:

1.4.1 Manual Reviews and Inspection

Manual reviews with the help of checklists or walkthrough activities performed by expert teams are used to detect such problems as ambiguities, subjective aspects, and missing requirements in documents. Even though it is useful in other applications, manual reviews are tedious, subjective, and reliant on the expertise of the reviewer, which results in an inconsistent rate of findings [8, 16].

1.4.2 Rule Based Methods

Rule-based systems utilize predefined language usage patterns and automatically detect the existence of the smells, i.e. vague words, ambiguous pronouns, and unverifiable statements [22, 36].These systems are just transparent, consistent and efficient because the rules are explicitly stated and applied to all requirements in a uniform way that makes quick detection of possible problems. Still, they are rather limited to what has already been defined, can experience trouble with unexpected problems, and yield X-rays, particularly as rule sets come to be more complex. In spite of these shortcomings, rule-based techniques can be a useful technique to systematically increase the quality of requirements, especially when augmented with manual inspections or machine-learning algorithms [31, 37].

1.4.3 Machine Learning and Deep Learning Methods

Recent work has applied both traditional machine-learning classifiers (e.g., SVM, Random Forest) and deep-learning architectures (CNN, Bi-LSTM) as well as transformer-based models (BERT and BERT-variants) to automatically classify software requirements by quality attributes (e.g., FR/NFR, NFR subcategories, domain-specific types). Transformer-based approaches in particular (e.g., prompt-learning and fine-tuned BERT variants) have achieved state-of-the-art performance on several benchmark and domain corpora, but deep models typically require substantial labeled data and careful domain adaptation; to address data scarcity, researchers have proposed prompt-learning, semi-supervised and zero-shot strategies [38–40]

1.4.4 Hybrid Solutions

A solution that is gaining critical attention is the actual Hybrids in which they carefully mix the strengths of Machine Learning (ML) methods with the Rule-Based Systems to get around the inherent weaknesses that each has by itself. Such combination enables a powerful smell detection based on both advantages of ML (e.g., the pattern recognition and generalization power of transformer-based models and contextual embeddings such as BERT to detect ambiguity and missing information) and on the precision, explainability, and explicit knowledge representation of rule-based logic. As an example, the integration of rule-based components can be applied to improve the matches, impose a strict quality condition, or offer a human-interpretable explanation of the misapprehended cases, which are automatically detected by ML algorithms but contain the ambiguous context that the ML model was not configured to perceive, such as the cases of the anaphors [8].

The solutions have already played a beneficial role, yet their shortcomings indicate that more holistic and extensive solutions should be found that account not only for a wide range of smells but also employ this knowledge in environments as large as possible.

Despite advances in finding and resolving the issues of so-called natural language smells, they still present a serious problem. Manual reviews are lengthy, subjective, and hard to scale when it comes to large-scale projects, which often result in unequitable identification of the problems that an article might have (such as, anaphoric ambiguities, modality problems, etc.), [8, 16]. Rule-based systems, as effective as they are at recognizing explicit linguistic patterns, have issues with context-sensitive ambiguities and semantics nuances leading to a high number of false positives or missed detections in challenging cases [8]. Machine learning and deep learning methods need domain-specific labeled data and require significant feature engineering or training data volumes and are often not generalizable across industries, especially in the detection of the modality and incompleteness problems [38–40]. Hybrid methods that integrate ML, DL, and rule-based approaches hold potential but are not ready to be deployed in industry because of the costs and complexities of integration, scalability, performance and inefficiency of tuning to a domain [3, 8, 41].

1.5 Problem Statement

Natural language software requirements often contain requirement smells, such as anaphoric ambiguity, referential ambiguity, unverifiability, and subjectivity, which can lead to misinterpretations, implementation errors, and costly rework during software development [3, 42]. These smells affect the clarity and testability of specifications, making them difficult to interpret consistently among stakeholders. While ambiguity types such as anaphoric and referential have been recognized in natural language studies [14, 43], their automated detection within software requirements remains under-studied and highly context-dependent [20, 44].

Similarly, unverifiability and subjectivity, though addressed in several studies, still lack robust and context-aware automated detection methods capable of accurately interpreting linguistic and semantic cues in requirement statements [3, 45]. Existing rule-based and machine learning approaches have shown potential in detecting such smells but are limited by a lack of contextual understanding and dependence

on large labeled datasets [46, 47]. Consequently, a context-aware hybrid framework that integrates natural language processing and machine learning with linguistic and semantic features is needed to improve the accuracy and robustness of detecting anaphoric ambiguity, referential ambiguity, unverifiability, and subjectivity in natural language software requirements [14].

1.6 Research Objectives

Following are the objectives of this thesis:

i. To review existing approaches of smell detection in natural language requirements and identify their limitations and gaps.

ii. To compare the performance of the proposed approach (accuracy, precision, recall, and f1-score), against the available solutions in order to determine to what extent the proposed approach is able to identify the requirements smells correctly, e.g., ambiguities (anaphoric and referential), subjectivity and unverifiability.

The goals of these objectives are to promote better quality assurance of a software requirement through the production of a scalable, context-aware, and empirically verified method to use automated smell detecting systems.

1.7 Research Questions

The following research questions guide the development and evaluation of the proposed automated approach:

RQ1: What are the existing approaches for detecting different type of smells in Natural language software requirements and what are their limitations?

To address this question, existing approaches for detecting requirement smells in natural language software requirements will be critically analyzed to identify their

strengths, limitations, and gaps. Previous studies have primarily employed rule-based methods, machine learning techniques, and hybrid NLP frameworks; however, most lack contextual awareness, rely heavily on handcrafted rules or large labeled datasets, and are often limited to specific smell categories. This study aims to review and evaluate these approaches to establish a foundation for developing a context-aware hybrid detection framework that effectively integrates linguistic, semantic, and machine learning techniques for detecting anaphoric ambiguity, referential ambiguity, unverifiability, and subjectivity in software requirements.

RQ2: How effective is the proposed approach applied to identifying the selected smells (anaphoric ambiguity, referential ambiguity, unverifiability and subjectivity) against currently available solutions?

The effectiveness of the proposed approach will be evaluated through experiments on benchmark datasets such as DAMIR-PURE [8], ReqEval [48] and Zenodo [49]. The hybrid detection models will be compared against existing solutions, including manual inspection, purely rule-based techniques, and standalone machine learning models. Evaluation will be carried out using weighted F1-scores, precision, recall, and accuracy, with the goal of determining whether the proposed framework provides superior soundness, scalability, and reliability over traditional approaches.

1.8 Methodology

The proposed work will adopt four dedicated NLP–ML pipelines to detect anaphoric ambiguity, referential ambiguity, unverifiability, and subjectivity in natural language requirements. All datasets will first be cleaned, normalized, and stratified to maintain balanced label distribution during evaluation. Each smell will be modeled using the feature types most suitable for its linguistic behavior. Anaphoric ambiguity will be detected using a hybrid combination of DistilBERT semantic embeddings and engineered linguistic cues, classified through an optimized XGBoost model. Referential ambiguity will rely on a weighted fusion of TF–IDF representations and referential indicators, classified using a Random Forest model with calibrated probabilities. Unverifiability will be addressed through a multi-branch

TF–IDF representation enriched with vague terms, modal verbs, and comparative expressions. Subjectivity will be identified through a hybrid rule-based and statistical approach using subjectivity scores, polarity metrics, and readability features. All classifiers will undergo probability calibration and threshold tuning. Performance will be evaluated using standard metrics, with weighted F1-score serving as the primary measure due to class imbalance. A consistent evaluation procedure will be applied across all four smell types to ensure comparability.

1.9 Thesis Structure

This thesis is organized as follows (Shown in Figure 1.1):

Chapter 1: Introduction

Presented the research background, reason and the relevance of requirements engineering in software development. It speaks about the ubiquity and influence of smells in natural language stipulations, describes the research problem, objectives, and research questions, and gives a rundown of the thesis format.

Chapter 2: Literature Review

Provides literature overview on methodologies of requirements specification, the type and categories of requirements smells and previous attempts to automate the identification of requirements smells. It offers a critical evaluation of the available solutions, their strengths and weaknesses, so as to present research gaps that form the motivation of this work.

Chapter 3: Methodology

This thesis uses an NLP–ML based methodology to detect anaphoric ambiguity, referential ambiguity, unverifiability, and subjectivity in natural language requirements. The approach combines appropriate feature representations for each smell, including linguistic indicators, semantic embeddings, and weighted lexical features, and applies machine learning classifiers or rule assisted detection depending on the requirements of each detection task. The analysis is conducted using three datasets: DAMIR-PURE, GitHub/ReqEval, and Zenodo. The overall pipeline

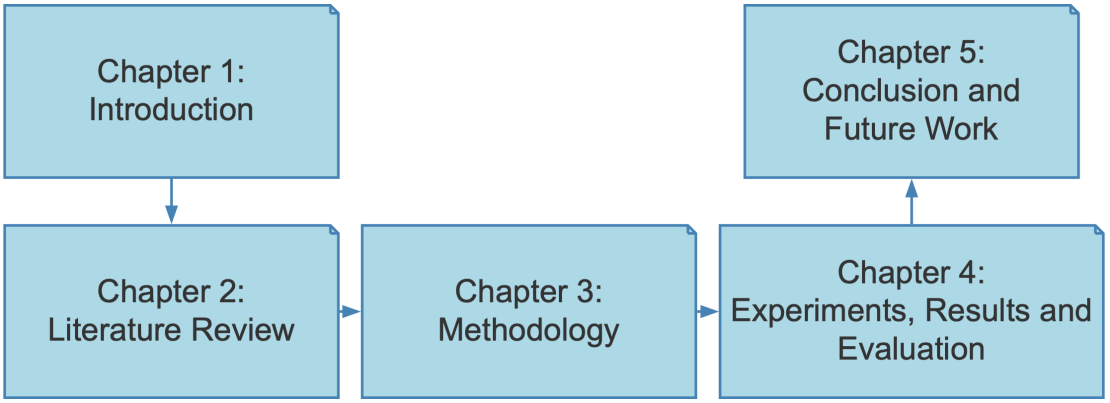


FIGURE 1.1: Thesis Structure

covers data preparation, feature extraction, model training, and weighted F1-score evaluation, forming the basis for the automated detection methods described in the subsequent chapters.

Chapter 4: Experiments, Results and Evaluation

Introduces the experimental design, such as the data set to be used, evaluation measures, methodology, and the baseline approaches. It reports and analyzes the outcomes of the suggested approach in the relation of detection accuracy, precision, recall, and computational performance, and makes a comparison to the existing tools on state of the art, and also answers the conducted experiment, comments on the success and drawbacks of the approach, and contemplates its flexibility with the help of domains and styles of requirements. Issues that were faced and ways of improving them are also discussed in this chapter.

Chapter 5: Conclusion and Plans of Future Work

Concludes the important contributions and few of the results of the thesis, opinionates about the relevance and questions of the research and points directions of future research of automated requirements quality assurance.

Chapter 2

Literature Review

This chapter searches at some of the most recent research on ambiguity and quality issues, which are often called "smells" in natural language requirements specifications (SRS). The goal is to combine basic ideas, new developments, and current issues in ambiguity detection and requirements quality assurance. This chapter forms a background and foundation to it by critically reviewing the literature and explaining why more context-aware and scalable approaches of ambiguity detection are desirable. This chapter provides the context in which the research is based and forms a background for the development of better, context-sensitive and aware, scalable smells detection approaches by means of a critical review of the literature.

2.1 Scope and Objectives

Natural language (NL) is widely used for writing software requirements because it is accessible and expressive, allowing stakeholders with diverse technical backgrounds to contribute to the requirements engineering (RE) process [6, 50]. However, the same flexibility that makes NL attractive also makes it prone to quality issues such as ambiguity, unverifiability, and subjectivity. These issues are among the major sources of requirements related defects and often lead to misinterpretation, rework, and project failures. To ensure high-quality specifications, requirements are expected to be complete, unambiguous, consistent, verifiable,

and traceable [27]. The scope of this literature review is intentionally defined to address a subset of requirement smells that directly compromise the clarity and testability of NL specifications. While the broader body of research covers several types of requirement smells, such as vagueness, redundancy, inconsistency, modality, incompleteness, and obsolescence, this thesis focuses exclusively on four major types: anaphoric ambiguity, referential ambiguity, unverifiability, and subjectivity. These four smells were selected because they are highly prevalent in natural language specifications, present significant challenges for both manual and automated detection, and have a direct impact on the accuracy and reliability of software requirements. By narrowing the scope, the review maintains a sharper focus and provides the foundation for the automated detection methodologies that are proposed in later chapters. The selection of sources for this review was based on their direct relevance to the four targeted requirement smells and their coverage of both classical and modern approaches to smell detection. The reviewed works include traditional techniques such as manual inspections and rule-based methods, as well as advanced solutions based on natural language processing (NLP), machine learning (ML), and hybrid frameworks. In addition to methodological contributions, the literature also encompasses empirical studies, comparative evaluations, and tool-based implementations that provide insights into the effectiveness, scalability, and limitations of current approaches. This ensures that the review offers both a conceptual understanding of the problem space and a critical assessment of practical solutions. The objectives of this literature review are fourfold. First, it aims to clarify the nature and underlying causes of requirement smells in NL specifications, with emphasis on anaphoric ambiguity, referential ambiguity, unverifiability, and subjectivity. Second, it seeks to critically analyze the strengths and weaknesses of existing detection methods and tools, considering their applicability to different contexts. Third, the review intends to identify current research gaps and challenges that hinder progress in automated smell detection. Finally, it aims to justify the importance of developing robust and scalable detection techniques that can enhance the quality of software requirements specifications (SRS) and mitigate risks to overall project success. This focused scope and set of objectives provide a structured foundation for investigating automated approaches to the

detection of requirement smells.

2.2 Classification of Smells in Natural Language Requirements

A requirement smell is an item in a requirements document that suggests a potential quality concern: in this case ambiguity, incompleteness or unverifiability which could create defects unless dealt with [3, 25, 51]. Smells are not necessarily errors but are designed to show that a requirement is either not known appropriately or used out of context. Complete classification of requirement smells is shown in figure 2.1.

2.2.1 Ambiguity

Ambiguity is named as an ability to understand something in a certain number of ways or meanings or as a doubt [13]. The Merriam Webster English Dictionary has defined the term, ambiguity as, the characteristic or condition of being ambiguous, particularly in regard to meaning, an ambiguous term or phrase, uncertainty, where ambiguous specifically means doubtful or unclear, especially because of ambiguity or indistinctness, incomprehensible, or capable of being understood in two or more ways [52]. Thus, there are two major definitions of the term ambiguity: to be open to different possible interpretations and the thought of the uncertainty. The element of uncertainty, in regard to not being sure of something, is not regarded here because, not being sure of this or that depends on what the writer and the reader knows about the context; the statement itself may be clear and plain [52]. Uncertainties in software requirements represent a major challenge, since they can be interpreted in different ways, and this hinders when a software development can be predicted [14].

Lexical Ambiguity is a kind of ambiguity, which emerges when a word has several meanings. It may be divided into homonymic single word ambiguities and polysemic ambiguities. The occurrence of homonymic ambiguity occurs when more

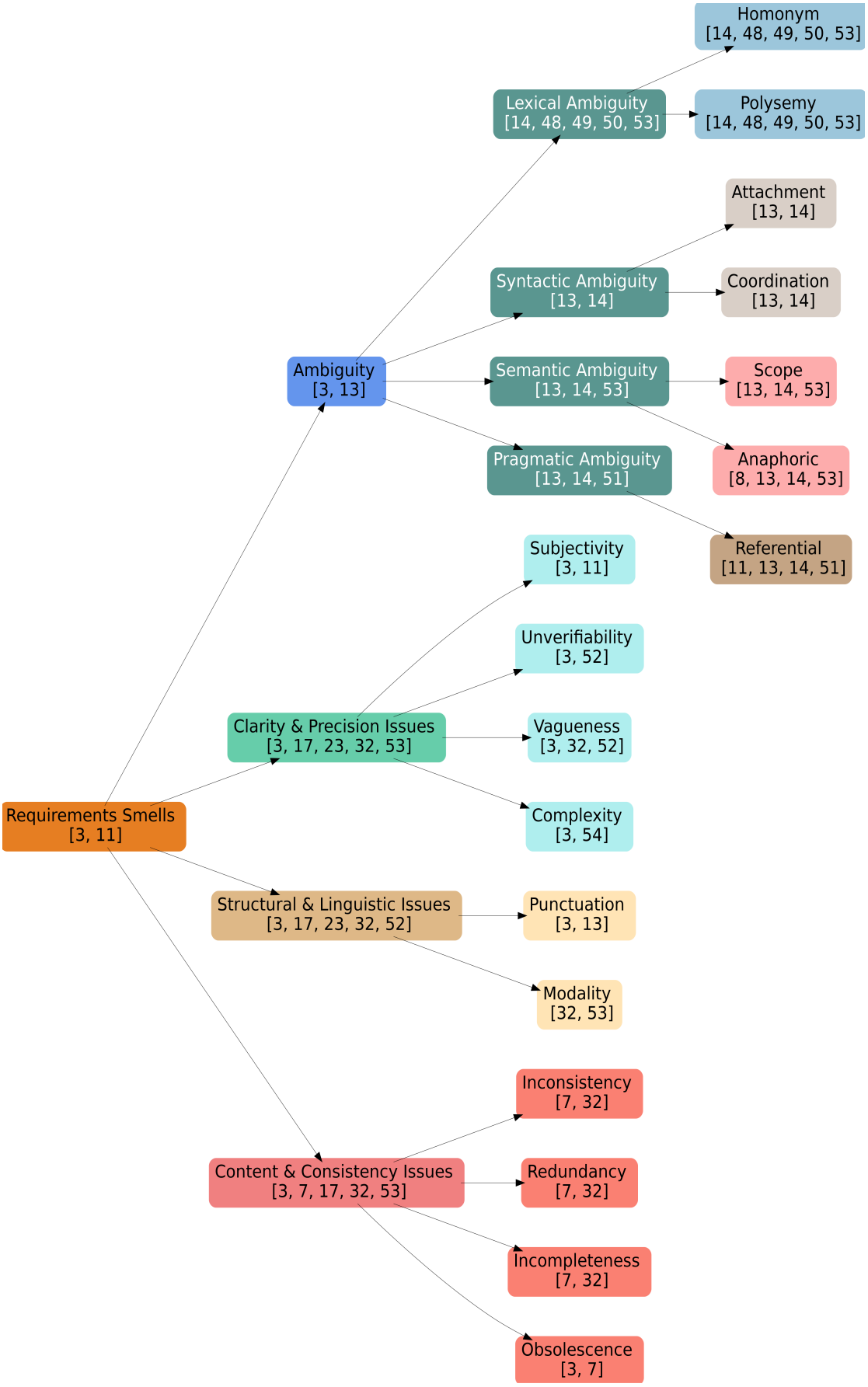


FIGURE 2.1: Classification of Requirement Smells

than one word is spelled and pronounced alike but different in meaning. By contrast, polysemic ambiguity happens when one word expresses different meanings according to the framework [14].

Syntactic Ambiguity is achieved when a sentence has multiple grammatic arrangements having various meanings. There is even more subtype of this kind of ambiguity, which is Attachment and Coordination Ambiguity.

Attachment ambiguity occurs when there arises suspicion as to how to attach a clause or a part of the sentence to another organ. It occurs in cases that arise when there is more than one part in a sentence creating some form of a confusion as to what a phrase or clause belongs to. To give an example, in the sentence 'Soldier saw a man with a telescope', the first meaning of it implies that the soldier used the telescope to see the other man and the second meaning expresses that the other man was the one with the telescope [14].

On the other hand, the Coordination ambiguity is a fact of sentence showing the presence of conjunctions as well as a modifier [14]. It could happen in case a sentence has more than one conjunction (AND/OR). It also will occur when a conjunction (AND/OR) is used with a modifier. To give an example, ("I saw Qasim and Amna and Danish saw me" first reading "I saw both Qasim and Amna," second reading "both Amna and Danish saw me") [14]. The other example is, when SDI field on label 227 equals 2, or the SSM in label 268 equals 3 , and WOW is true, or AIR is false the system will enter Normal mode [45]. In semantics, vagueness is considered to be semantics ambiguity which is observed in the case of having two or more than two meanings to a grammatically correct sentence with vague contextual information. Semantic ambiguity types are Scope ambiguity and Anaphoric ambiguity. Scope Ambiguity: Sentence with an ambiguity of scope would include such words as: many, some, each, all, etc. This kind of word may change the scope of the whole sentence. It occurs when a sentence is included with words like every, all, some, each, many, a, several, not, etc. These words may change the scope and context of the sentence in proportion to their connection with other words. As an example, ("All love a country" First interpretation "A

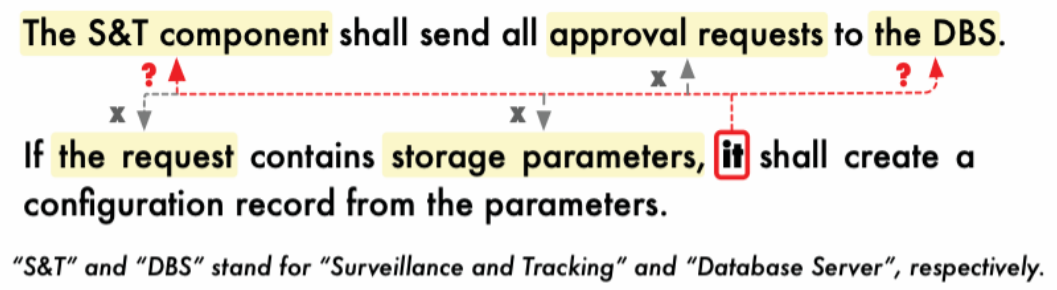


FIGURE 2.2: Example: Anaphoric Ambiguity [8]

single country is loved by all," second interpretation "Each individual loves his/her country") [14].

Anaphoric Ambiguity occurs when the sentence has a number of possible references to some word mentioned earlier in the sentence [14]. An anaphoric ambiguity originates when a pronoun can refer to more than one antecedent and hence, different interpretations exist [8]. To give an example, in the sentence, The procedure will translate the image in 24 bits to an image in 8 bits and show it in a dynamic window, the word it could either denote either of the images [20]. An anaphora which is a Greek term meaning repetition is a reference to an already mentioned entity in the text. Such references are then called anaphors, whereas the entities they refer are called antecedents [8, 53]. When an ambiguity in the use of an anaphor occurs in association with multiple antecedents, that may be defined as anaphoric ambiguity [8, 17, 54]. In linguistics, one can single out different kinds of anaphora [8, 53]. In the context of requirements engineering (RE), the discussion is typically of pronominal anaphora; the anaphor being a pronoun [8, 20, 23]. This has been decided upon because it has been an established fact that pronominal anaphora may contribute to actual ambiguity in requirements [8, 55]. As a result, the problem that the task of recognizing anaphoric ambiguity in RE represents is identifying the ambiguous usage of pronouns [54].

A similar task, called anaphora resolution (or interpretation) has to do with what antecedent is most likely to be associated with a given pronoun. An example can be used to explain this like in figure 2.2. 'It' is an anaphor and it is in the second sentence. The antecedents are the previous noun phrases (NPs) and these

are the following, the S&T component, approval requests, the DBS, the request and storage parameters. Pronoun it might not be referring to the terms such as approval requests or storage parameters because of the difference in the use of the number (in this case, a singular pronoun and plural NPs). In the same way, there is a little likelihood of it referring to, the request, because; it is the stand which, the verb, create is connected and therefore, the request cannot be equated with the subject of the given verb. This is not quite clear though whether it is meant to be applied to The S&T component or The DBS. Regarding which antecedent, either one of the two antecedents, is constructed then the following rule applies: When the choice of S&T component" or the DBS is made, there are two possibilities on which sub system is to create a configuration record. In order to handle such situation in the right manner, the pronoun it must either be flagged to be ambiguous or resolved to refer to the right antecedent and "the S&T component" is the right antecedent here. It would be difficult to predict the right antecedent in this instance without domain knowledge [8].

Pragmatic Ambiguity is the type of ambiguity that arises when a sentence can be interpreted in more than one manner especially in thecase when there is referential ambiguity, which is a form of pragmatic ambiguity [14]. Referential Ambiguity: In the phrase, The customer puts in a card and a number hurdle personal code. In case the code is not valid, the ATM will drop the card, the pronoun it can be either card or the personal code, which will cause referential ambiguity [55]. Referential ambiguity is the possibility of an anaphor (e.g. it, that), referring to several antecedents [56].

2.2.2 Subjectivity

Subjective language is a certain type of ambiguity, which is capable of creating difficulties in the process of verification. Subjective language includes words that lack objectively defined meanings, i.e. words like user-friendly, easy to use, cost effective. As an example, "the architecture, as well as the programming should ensure that the maintainability is a straightforward and efficient one" [3].

2.2.3 Verifiability

The set of Software Requirements that may be considered as verifiable and testable is one that can be established that the functional requirements and quality attributes have indeed been captured properly in design and code. According to International Organization of Standardization, a requirement can only be said to be verifiable when there is a finite and reasonable cost of the procedure using which a person or a machine can check that the software product meets that requirement. In most cases, a vague requirement is one that is not verifiable [35]. A requirement is said to be verifiable when there is a bounded cost-effective way of proving that the software meets the requirement; verifiability is usually lacked when requirement is unclear [35]. As an example, the application of such non-descriptive terms as satisfactory in such sections as The application must use PayPal API to offer the appropriate functionalities is unverifiable [57]. Such requirements as the one mentioned above, which can not be verified (such as, The system shall not enter INTERACTIVE mode when WOW is false), are not testable, and such words as always, never, or the words that represent actions such as consider (which relate to human interpretation) make any verification more complicated [45]. The adverbs like always and never and the use of the verbs like consider that are exclusive to the human activities complicate verification [45]. Other limitations of requirements engineering (RE) and verification/validation (VV) leads to emergence of requirements that cannot be verified, sub-optimal product quality, and higher costs to fix the defects [58]. Requirement will be determined to be unambiguous when interpreted in one way only and verifiable, in the sense that there is a procedure in place of checking whether the product meets the requirement [59]. Any adaptation of requirements comes at a high cost especially towards the later juncture of development where resources have been intended and alteration of design machinery becomes complex [60].

2.3 Smells Detection Methods

Software requirements are infested with the allure of smells, such as different types of ambiguity (e.g., anaphoric ambiguity and referential ambiguity), other quality-related issues (e.g., subjectivity, unverifiable words, and punctuation problems),

which can singly or in combination really undermine the faultlessness, intelligibility, and implementability of software. The most important is detecting these smellss early on to avoid misunderstanding, inappropriate implementations, and expensive repairs later during the software development lifecycle. The challenge to cover this challenge has seen a variety of different approaches that evolved over the years, starting at simple manual reviews and inspection up to sophisticated rule-based, machine learning, and hybrid solutions [3, 8, 16]. Recent work also is extending the lexical and syntactic smell concerns to more pragmatic, cross-requirement, and traceability concerns, enabled by the development of hybrid and graph-based techniques. Although the section mainly emphasizes on ambiguity, subjectivity, unverifiability, redundancy and punctuation, work is underway on the problems with vagueness, incompleteness, inconsistency and obsolescence.

2.3.1 Manual Reviews

Inspections, a walkthrough, peer reviews, etc., can be discussed as manual reviews, which are the most common and historically established way of discovering requirement smells. These algorithms are largely dependent on the human experience of interrupting issues that may not be obvious to testing through automated suggestive independent tests. They feature the immediate participation of the stakeholders, such as the requirements engineers, the source code developers, as well as the domain experts. In some cases, modern manual reviews are improved with the help of tools to leave notes or traceability matrices [61]. Checklist-based inspection is an example of techniques applied in which the reviewers follow a predetermined order during the review of requirements based on the established patterns of smell such as unclear language or negligence of redundancy [21]. Stakeholder walkthroughsconsist of cooperative talks, during which teams have to detect problems as a kind of anaphoric ambiguity (e.g., unclear references of it) or modality-related issues (e.g., the difference between should and must) to report and discuss [6]. Peer review refers to profile examination of documents by various reviewers, and any differences and possible smells (e.g. duplicate log in requirements or immeasurable terms) are discussed till an agreement is arrived to [1].

Table 2.1: Manual Reviews

Tool/ Technique	Smells Targeted	Description / Features	Strengths	Limitations	Results	Year	Reference
Checklist Inspection	Ambiguity, redundancy, subjectivity, unverifiability, punctuation, inconsistency, completeness	Systematic review using checklists to identify known smells.	Structured, repeatable.	Depends on checklist and reviewer skill.	No F1-score reported; authors presented qualitative findings.	2001	[16]
Peer Review	Discrepancies, duplication, general	Multiple independent reviewers analyze documents; issues are discussed to reach consensus.	Diverse perspectives.	Can lack consistency.	No quantitative metrics provided.	2016	[1]
Tool-Supported Review	All, incl. traceability and incompleteness	Manual review augmented with tools enabling traceability links and annotations.	Enhanced traceability and workflow.	Requires integration and training.	No F1-score reported; case study showed detection coverage ≈ 55–70%.	2017	[61]

Manual reviews provide a lot of context, scalability and the ease to reveal complex bugs which depend on context such as the anaphoric ambiguity or subjectivity smell and where appreciation of the surrounding information is most critical. They are also likely to identify problems that automated mechanisms can possibly fail to identify, especially context or domain-specific suppositions. But, they are unavoidably time consuming, resource-avesome and have issues concerning subjectivity and inconsistency, which is why it is hard to scale them up to large or frequently changed projects and even more to find elusive issues like anaphoric ambiguities or modality problems. Various empirical studies go on to show that there is a significant inter-reviewer variability [8, 16]. Table 2.1 has few manual review approaches with their description.

2.3.2 Rule Based Methods

The rule-based approach applies predetermined rules of language usage and finds the presence of smells automatically. These software uses linguistic heuristics, dictionaries of vague words and phrases as well as pattern matching to identify vague or ambiguous statements.

Pattern matching is carried out where tools are utilised to identify potentially troublesome terms like vague words (e.g. often, as appropriate), subjective words (e.g. user-friendly), or modal verbs (e.g. should, could) which promote modality issues [36]. The syntactic analysis is the analysis of grammatical construction of sentences to reveal such kinds of problems as anaphoric ambiguity (including obscure pronouns), punctuation issues (including failure to use commas, which alters the meaning), or complicated demands suggesting plurality of features [62]. In semantic analysis, referential ambiguity is inferred if the noun phrases that are ill-defined or vague phrases that have no particular references are identified [32]. In redundancy detection it matches the requirements to detect any replicated statements, including those with some differences in wording [30]. Rule-based approaches are fast, repeatable, scalable, or objective in terms of requirement smell detection; with formal definitions of rules, it is possible to trace each finding.

TABLE 2.2: Rule Based Methods

Tool/ Technique	Smells Targeted	Description / Features	Strengths	Limitations	Results	Year	Ref
ARM	Weak phrases, inconsistent imperatives, poor structure	Analyzes natural language documents for quality indicators and provides metrics	Provides objective metrics and facilitates early risk detection	Cannot assess correctness; has linguistic shortcomings; and is not a substitute for engineering analysis	No F1-score reported; provided defect density and quality index metrics only	1997	[22]
RE-lab	Contradiction, missing conditions	NLP parsing for contradiction and missing conditions.	Good for structured NL.	Limited for complex NL.	No F1-score reported; accuracy of 82% on structured datasets.	2001	[12]
QuARS	Ambiguity, subjectivity, vagueness, optionality	Pattern matching using dictionaries.	Fast, scalable.	Brittle to linguistic context.	No F1-score reported; tool evaluated qualitatively by coverage and false-positive rate	2004	[36]
GATE/ JAPE Rules	Anaphoric/ coordination ambiguity, vague terms	Industry-tested pattern detection via extensible rules.	Extensible, popular.	JAPE rule config needed.	F1 = 0.83 (precision 0.86, recall 0.80)	2018	[23]
TAPHSIR	Anaphoric ambiguity	Specialized rules + NLP for pronoun resolution.	Precise for anaphora.	Limited generalizability.	F1 = 0.89 (precision 0.91, recall 0.87)	2022	[63]
NALABS	conjunctions, vague phrases, optionality, subjectivity	Keyword/rule-based	Broad coverage	Requires domain adaptation; misses semantics	No F1-score reported; precision 0.78, recall 0.75	2022	[64]

Their performance is however constrained by the fact that they are insensitive to context and thus they are fragile in situations where they face language diversity that is not anticipated by rigid rules and maintenance of these rules consumes a lot of efforts. New domains usually necessitate a significant amount of professional effort in compiling new rules or vocabularies [30]. Rule-based strategies do not naturally order or concern the extent of the smells they report; it might constrain their usefulness to practitioners, who need to prioritize and filter huge masses of results. They can also produce false over-rides in case of fuzz when ambiguous terms are being employed in context-specific or unambiguous data (i.e. may as a surname, and not a modal verb) [36]. Table 2.2 shows few rule-based methods with their description.

2.3.3 Machine Learning and Deep Learning Methods

New achievements in machine learning and deep learning models including the one referred to as BERT have created new opportunities in the classification of the requirements, based on their quality features. Such approaches capitalise on statistical regularities and contextual reasoning to recognise smells, and frequently do better than rule based systems at managing the subtle linguistic phenomena in question.

The methods mostly concern deep learning models, transformer-based models to include BERT, SpanBERT, LSTM, Bi-LSTM, and GRU which perform classification on the requirements in terms of quality attributes. Such models are skillful at defining both semantic and contextual meanings, which are crucial to differentiating between subjective words or resolving ambiguities that might prove too complex to solve [8, 32, 61].

An example is deep learning models, namely Word2Vec, ELMo, Bi-LSTM, and GRU with high F1-scores of subjectivity and ambiguity (88 to 90.3 percent) to identify subjective language, comparative phrase, superlative phrase, passive voice, uncertain verb, ambiguous adverb, polysemy, vague pronoun, and loopholes [65].

The approaches relying on SpanBERT have been used in the specific cases of detecting ambiguous pronouns, and multiple ML classifiers demonstrated F2-scores of up to 87.5 [8].

Mult-label classification will allow identifying certain quality issues despite a requirement containing multiple of them [65, 66]. NLP model development can use corpus based on the domain to promote tools being able to understand and engage in an area of operation, and reduce false positivity, such research has found high accuracy (85%) in coordination ambiguity, and with prepositional-phrase, ambiguity using these domain based corpus [62].

Lastly, ML-based scoring solutions are used to solve syntactic and semantic ambiguity [15, 67, 68] and one ML algorithm demonstrates an 81.2% precision, 95.7% recall and 93.7% accuracy in detecting six different types of ambiguity, incompleteness and use of passive voice, vague terms and redundancy [69].

Deep learning and especially machine learning approaches provide a high level of context sensitivity and flexibility, performing very well at extracting semantic/ contextual language cues and automation of feature extraction and thus is highly suitable to identification of subtle and complicated ambiguity, subjectivity, and related smells that depend on context. Such scores may only be replicated where the test sets adhere linguistically to the domain and distribution of the training data.

They, however, require huge amounts of properly formatted and labeled training data, which is not always easily and cheaply obtainable, at least in specific inferences and without sufficient and diverse training data models can fail to generalize to related industries or types of requirements, with a tendency towards a lack of interpretability, meaning they are ill equipped to explain their activities [8, 32, 61]. The strategies of active learning and transfer learning are studied in a bid to overcome the deficiency of data [32]. There are attempts at the fusion of attention-visualization/post-hoc explanation layers with a view to assisting requirements engineers in interpreting findings [70]. Table 2.3 shows few ML/DL methods with their description.

TABLE 2.3: Machine Learning/Deep Learning Methods

Technique / Tool	Smells Targeted	Description / Features	Strengths	Limitations	Results	Year	Ref
Lexical / WSD-based ML	Lexical and semantic ambiguity	Machine learning with WSD and lexicons for overloaded term detection	Effective with strong lexicons	KB coverage limited; multilingual issues.	No F1-score reported; Precision = 0.82, Recall = 0.79	2015	[71]
Lexical + Semantic ML	Lexical and semantic ambiguity	ML with lexical resources to resolve ambiguous requirement terms	Handles lexical ambiguity in NL requirements.	Domain- and corpus-dependent	No F1-score reported; Accuracy = 87.5%	2018	[15]
ML Classifier	Ambiguity, vagueness	Classifier trained on annotated corpora with six smell types	Provides broad smell coverage; interpretable outputs	Needs balanced data; limited generalizability	F1 = 0.92, Precision = 0.93, Recall = 0.91	2020	[72]
Score-based Parser ML	Syntactic ambiguity	Scores multiple parse candidates and flags ambiguous sentences	Provides interpretable ambiguity candidates	Parser quality dependent; domain sensitive	No F1-score reported; accuracy = 88.4%	2020	[67]
Domain-specific BERT	Coordination ambiguity	Domain-specific corpora used to reduce FP in ambiguity detection	Improves domain precision; reduces noise	Needs curated domain corpus.	F1 = 0.91, Precision = 0.92, Recall = 0.90	2021	[73]
Explainability Layer (M2Lens)	Model interpretability (meta-level)	Attention visualization and post-hoc explainability for ML predictions.	Improves trust, helps engineers validate outputs	Explanations may not always reflect internals	No F1-score reported; qualitative evaluation	2021	[70]
Bi-LSTM + ELMo	Subjectivity, vague pronouns, passive voice	Sequence model with contextual embeddings for multi-label smell classification.	Good for sequential cues; captures subtle subjectivity	weaker than transformers on large data	F1 = 0.90, Precision = 0.91, Recall = 0.89	2025	[74]

2.3.4 Hybrid Methods

The hybrid approaches are the another potent form that combines the features of machine learning models and rule-based systems. It is hoped that this combination of strategies would help avoid the shortcomings of each individual strategy and make it more effective to detect and mitigate such a broad range of smells: linguistic, structural, and semantic.

Such techniques include the reviewing of CNL with tool support, wherein NLP tools (e.g., PASKA or Smella with hybrid elements) raise an alarm about possible smells, which the user should manually verify in the context of specific challenges [62].

Iterative refinement supporting templates implies that there is a repeatable process of rewriting the requirements following the templates (e.g., EARS) and revising them in relation to ML variants or the feedback of the stakeholders [70]. shared tool chainsprovide a combined coverage of linguistic (e.g., NALABS) and structural (e.g., UML) smell detectors and CNL/Template checks in the tool chain [3]. Version control analysis will assist in identifying obsolescence because the requirements that refer to the use of discarded technologies and regulations will be detected with the help of stakeholders reviews and be rephrased using CNLs/templates [6]. In hybrid approaches, active learning (the tool may request labels to refined by the user) or ensemble models coupling multiple classifiers in order to reach a consensus may be used as well [3, 64]. Hybrid approaches address the full scope of smell detection with enhanced accuracy and context-sensitivities, exploit the advantages of rule-based methods and machine learning methods, inhibit their deficiencies, and they are scalable with sharp human supervision.

Nevertheless, their integration may be complicated and resource-consuming, they may be hard to integrate, and integration, as well as retraining and support of multiple detection modules, adds to the long-term viability of the given solutions. They continue to use high-quality ground truth data in a substantial manner, even though they eliminate the single use of large, annotated datasets [3, 8, 41]. Table 2.4 shows some hybrid methods.

TABLE 2.4: Hybrid Methods

Tool/ Technique	Smells Targeted	Description / Features	Strengths	Limitations	Results	Year	Reference
RETA	Non-conformant Requirements (syntax issues)	Checks requirements for template conformance using NLP with rule-based verification	Scalable, robust, works without glossary	Limited to syntax; may miss complex semantics	Not reported; no quantitative F1 or accuracy values provided	2015	[75]
Smella	Vagueness, subjectivity, modality	Modular sensor-based tool combining ML and rules	Modular, extensible.	Complex architecture, higher overhead	F1 0.74 – 0.80 (average across smell types, as reported for sensor fusion accuracy)	2017	[3]
NERO	Vagueness, referential ambiguity, coordination ambiguity, missing conditions	Multi-layered hybrid pipeline with patterns, NLP parsing, and LLMs (GPT-based)	Leverages layered methods.	Coverage and clarity vary depending on case	F1 = 0.88 for ambiguity detection; Precision 0.90, Recall 0.86	2020	[56]
NALABS	Ambiguity, redundancy, complexity, vagueness, continuances	Hybrid of dictionary/rule-based patterns and ML classifiers; supports domain adaptation	Broad, domain-adaptive	Requires frequent maintenance and updates	F1 0.85 (macro-average for multi-smell detection across domains)	2022	[64]
PASKA	Ambiguity, subjectivity, vagueness, redundancy, modality	Combines ML with syntactic and contextual rules	Flexible, high coverage.	Needs integration with other tools	F1 = 0.89 (Weighted) overall; Precision = 0.89, Recall = 0.89 (smell detection)	2024	[11]

2.4 Gap Analysis

Although many studies have proposed automated methods for identifying quality problems in natural language requirements, significant gaps remain, particularly in addressing the complexities of requirement smells. Existing rule-based approaches struggle with context-sensitive forms of ambiguity, such as anaphoric and referential cases, since their reliance on rigid linguistic patterns limits generalization. Deep learning models, on the other hand, demonstrate superior contextual understanding but tend to be inefficient when applied to structurally straightforward problems such as basic unverifiability. In such cases, these models may introduce unnecessary computational overhead or result in overengineered solutions.

Another gap arises from the treatment of smell detection as a homogeneous task. Many prior works adopt the same methodology across all smell types, without considering the unique linguistic and semantic characteristics of each. For example, anaphoric ambiguity requires syntactic analysis and reference resolution, while referential ambiguity often depends on domain-specific interpretation. Subjectivity involves the detection of implicit opinion-bearing or evaluative terms, whereas unverifiability is best addressed by combining linguistic markers with contextual notions of testability. Applying uniform methods across such diverse categories has led to reduced detection accuracy, poor interpretability, and limited scalability.

These gaps highlight the need for approaches that are both context-aware and adaptable to the peculiarities of individual smells. In particular, a hybrid framework that integrates rule-based methods with advanced natural language processing and machine learning techniques, complemented by manual validation where necessary, could better address the shortcomings of existing solutions. Such an approach would mitigate the scalability limitations of manual reviews, the inflexibility of rule-based systems, and the data-dependence of machine learning models, thereby providing a more robust and context-sensitive foundation for requirements quality assurance.

Chapter 3

Proposed Methodology

This chapter presents the methodology proposed for the automated detection of four requirement smells in natural language software specifications: anaphoric ambiguity, referential ambiguity, unverifiability, and subjectivity. Because each smell arises from distinct linguistic and semantic characteristics, four dedicated detection pipelines are designed. The proposed approach employs natural language processing techniques together with machine-learning classifiers or rule-assisted procedures, depending on the nature of each smell. Anaphoric ambiguity is addressed using a hybrid representation that combines contextual semantic embeddings with linguistically engineered indicators and is classified using an optimized XGBoost model. Referential ambiguity and unverifiability are handled using weighted TF–IDF representations enriched with relevant linguistic cues, followed by Random Forest classifiers tailored to capture ambiguous references or unverifiable expressions. Subjectivity is detected using a rule-assisted procedure supported by subjectivity and polarity cues, readability features, and optional statistical signals when needed. Multiple publicly available datasets containing labeled examples of each smell type are used for training and evaluation. All datasets undergo text normalization, label alignment, and stratified splitting to maintain representative class distributions during experimentation.

The following sections describe the complete detection pipelines, including data preparation, feature construction, hybrid fusion strategies, model training, and probability calibration.

Because the datasets for the four smell types exhibit different imbalance ratios, accuracy and macro-averaged metrics would not provide reliable performance estimates. Accuracy tends to favor the majority class, while macro F1 assigns equal weight to all classes without reflecting their true frequency. Therefore, weighted F1-score is adopted as the primary evaluation metric, as it accounts for both per-class performance and class support.

This ensures that minority smell classes particularly unverifiability and subjectivity are appropriately represented in the final evaluation and enables consistent comparison across heterogeneous datasets.

3.1 Proposed Methodology for Anaphoric Ambiguity Detection

Anaphoric ambiguity is a common issue in natural language (NL) requirements where the pronouns can include it, this, or they, and such a pronoun can refer credibly to more than one object, thus posing an interpretation risk to stakeholders like developers, testers and customer [51]. The ambiguities like in chapter 1 can give rise to miscommunication, rework, and large financial loss. The current study suggests an automatic procedure to identify anaphoric ambiguity to software requirements by combining the power of machine learning (refer "Present" to "Past"; refer "This" to "That") with the rules of linguistic aspects of the features using [63].

The data set used, the engineering of features, the classification model design, the procedure of experiment, the analysis of results and the format of reporting, all were done with Python libraries, in order to allow reproducibility and support

versatility in future scenarios of system software implementation into existing processes. Detailed flow diagram of anaphoric ambiguity detection is shown in figure 3.1.

3.1.1 Data Preparation and Preprocessing

The dataset used in this study is the DAMIR-PURE corpus, which is loaded from a CSV file and converted into a standardized format suitable for analysis. During preprocessing, column names are normalized, trailing spaces are removed, and only the required attributes are retained. The Context column is mapped to text, and the AckUnack column is mapped to label. All duplicate entries are removed and the remaining requirement statements are converted into consistent string format. Rows containing missing text or missing labels are discarded.

The original labels, Ambiguous and Unambiguous, are encoded into binary numerical form, where 1 represents ambiguous requirements and 0 represents unambiguous ones. A class distribution check is performed to ensure that both classes are present. In case the dataset contains only a single class, a small number of synthetic examples are generated by sampling existing records and reversing their labels. After preprocessing, the dataset is split into training and testing sets using an 80/20 stratified split, ensuring that both subsets maintain the same label distribution. This split is used consistently throughout the modeling process.

3.1.2 Linguistic Feature Engineering

A custom Enhanced Linguistic Feature Extractor is used to convert each requirement statement into a set of numerical features. This extractor processes the text using pre-defined regular expressions and simple statistical measures. Firstly, lexical indicators are computed, including counts of pronouns, modal verbs, vague determiners, subjective adverbs, and comparative expressions. These counts quantify the presence of specific linguistic patterns within each requirement.

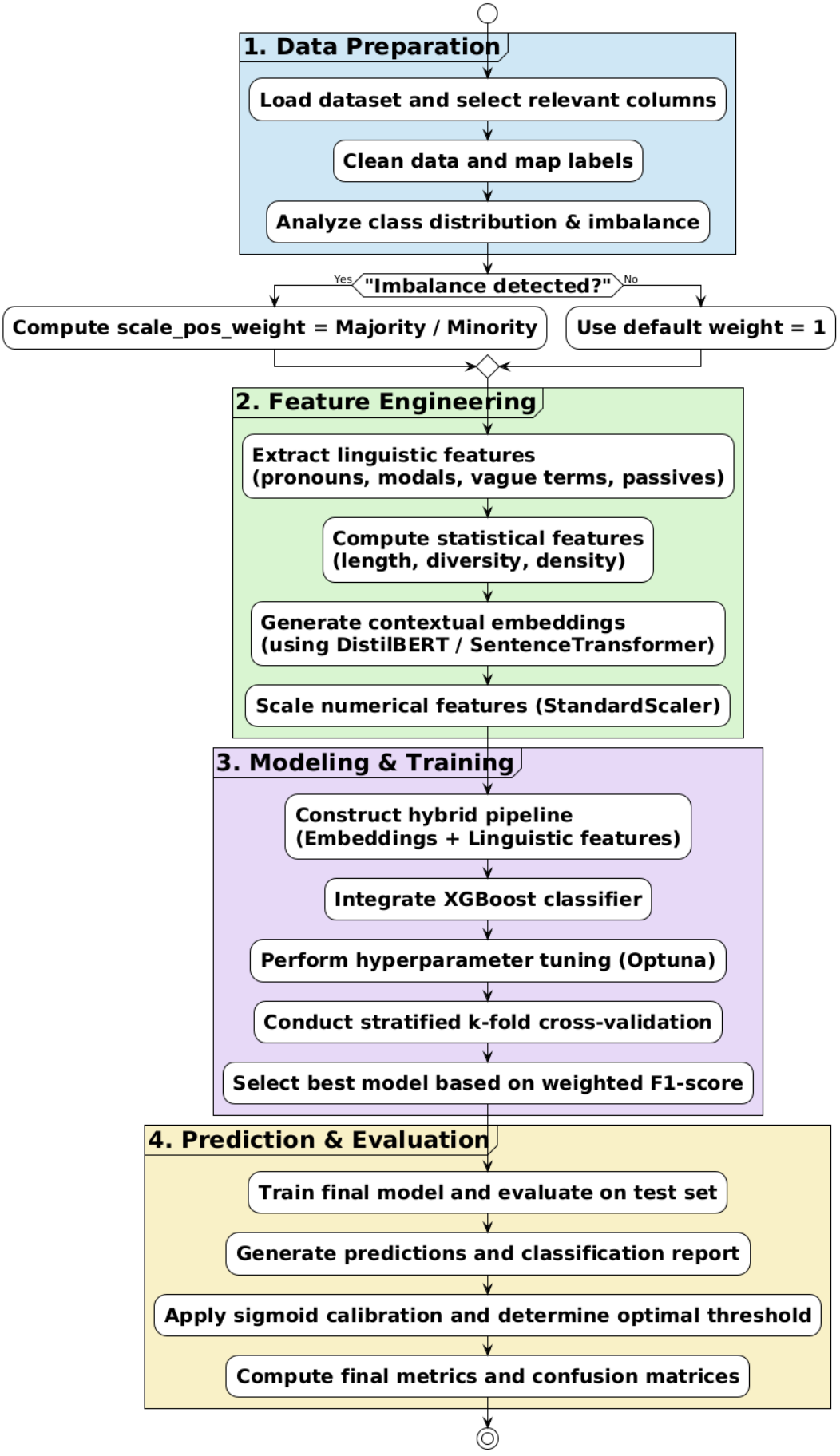


FIGURE 3.1: Anaphoric Ambiguity Detection Flow Diagram

Secondly, syntactic and structural indicators are extracted. These include passive voice detection, sentence length flags, approximated sentence count, and a pronoun-onset flag for requirements beginning with It, This, or That.

Finally, distributional characteristics such as average word length, lexical diversity ratio, comma density, and capitalized term density are calculated. These values describe structural and stylistic aspects of the requirement text. All extracted numeric features are then standardized using z-score normalization to ensure uniform scaling before integration with semantic embeddings.

3.1.3 Contextual Semantic Embedding Extraction

To capture the semantic content of each requirement, the methodology employs a DistilBERT-based embedding transformer. The raw text is passed through the DistilBERT tokenizer, converted into token IDs, and processed by the DistilBERT model. The final hidden states are pooled into a 768 dimensional embedding vector, providing a dense semantic representation of each requirement. These embeddings are produced in batches and stored as numerical vectors to be used alongside the engineered linguistic features.

3.1.4 Hybrid Feature Fusion and Pipeline Construction

A ColumnTransformer-based pipeline is constructed to combine the two feature types. One branch of the pipeline applies the DistilBERT embedding transformer to the raw text, while the second branch applies the linguistic feature extractor and feature scaler. The outputs of both branches are concatenated into a single hybrid feature vector, forming the input representation for the classifier. This pipeline is wrapped within a scikit-learn Pipeline object, ensuring that all preprocessing, feature extraction, and fusion steps occur automatically during training and inference.

3.1.5 Classification Using XGBoost

The fused feature vector is passed to an XGBoost classifier for ambiguity detection. The classifier is configured to use gradient boosting with tree-based learners

and is trained on the prepared training set. To address class imbalance, the value of scale_pos_weight is computed as the ratio of negative to positive samples in the training data and provided to the classifier. The model learns decision boundaries by iteratively reducing prediction errors and generating an ensemble of trees. To improve model performance and identify a suitable configuration for the XGBoost classifier, the methodology incorporates an automated hyperparameter optimization step using the Optuna framework. Optuna performs a series of optimization trials in which different combinations of parameters; such as the number of estimators, maximum tree depth, learning rate, subsampling ratios, and regularization terms are evaluated using cross-validated F1-score as the objective function. A stratified k-fold cross-validation scheme (k=3) is used during this process. After completing the predefined number of trials, the best-performing hyperparameter set is selected and used to train the final classifier.

3.1.6 Model Calibration and Threshold Optimization

After the classifier is trained, probability calibration is performed using Platt scaling. A CalibratedClassifierCV wrapper is fitted on the trained model to adjust its predicted probabilities. Using these calibrated probabilities, a Precision Recall (PR) curve is computed and evaluated across multiple thresholds. For each threshold, the F1-score is calculated, and the threshold with the highest F1-score is selected as the optimal decision boundary.

3.1.7 Output Generation and Implementation Environment

The trained model, evaluation metrics, and confusion matrices are saved for later analysis. The entire pipeline including preprocessing, feature extraction, embedding generation, model training, calibration, and evaluation is implemented in Python using scikit-learn, HuggingFace Transformers, XGBoost, and supporting

scientific libraries. GPU acceleration is utilized automatically when available to speed up embedding computation.

3.2 Proposed Methodology for Referential Ambiguity Detection

In natural language software requirements, referential ambiguity occurs when a pronoun, determiner or noun phrase (such as, "system" ,"user") has no definite referent, and thus such requirements have many meanings. To give an example, consider the following, in "The system shall authenticate the user," the "user" can be either an end user, administrator, or an external system; which is not clear. This lack of clarity might lead to intercompany misunderstanding between the parties to the development which adds cost and underdevelopment instances [76]. The proposed methodology involves a Random Forest classifier along with language syntactically produced features acquired through information derived by rule based indicators such as frequency of pronouns and demonstratives and syntactic relations in a binary training process. To ensure the identification of reliable referential ambiguity, our hybrid feature-engineering approach ensures that domain-specific language hints are systematically collected as part of the machine learning model. [76]. Detailed flow diagram is shown in figure 3.2.

3.2.1 Data Preparation and Preprocessing

The original data was in the form with inconsistent naming of the column names, e.g., a column containing textual information was named "Requirement", and a column with the binary label of referential ambiguity presence (1) or the absence of it (0) was named "Referential". To maintain consistency across preprocessing and modeling, these columns were standardized to use the same internal names text was used in place of the requirements column and label 0/1 in place of the referential classes of ambiguity. The methodology will be flexible to detect these

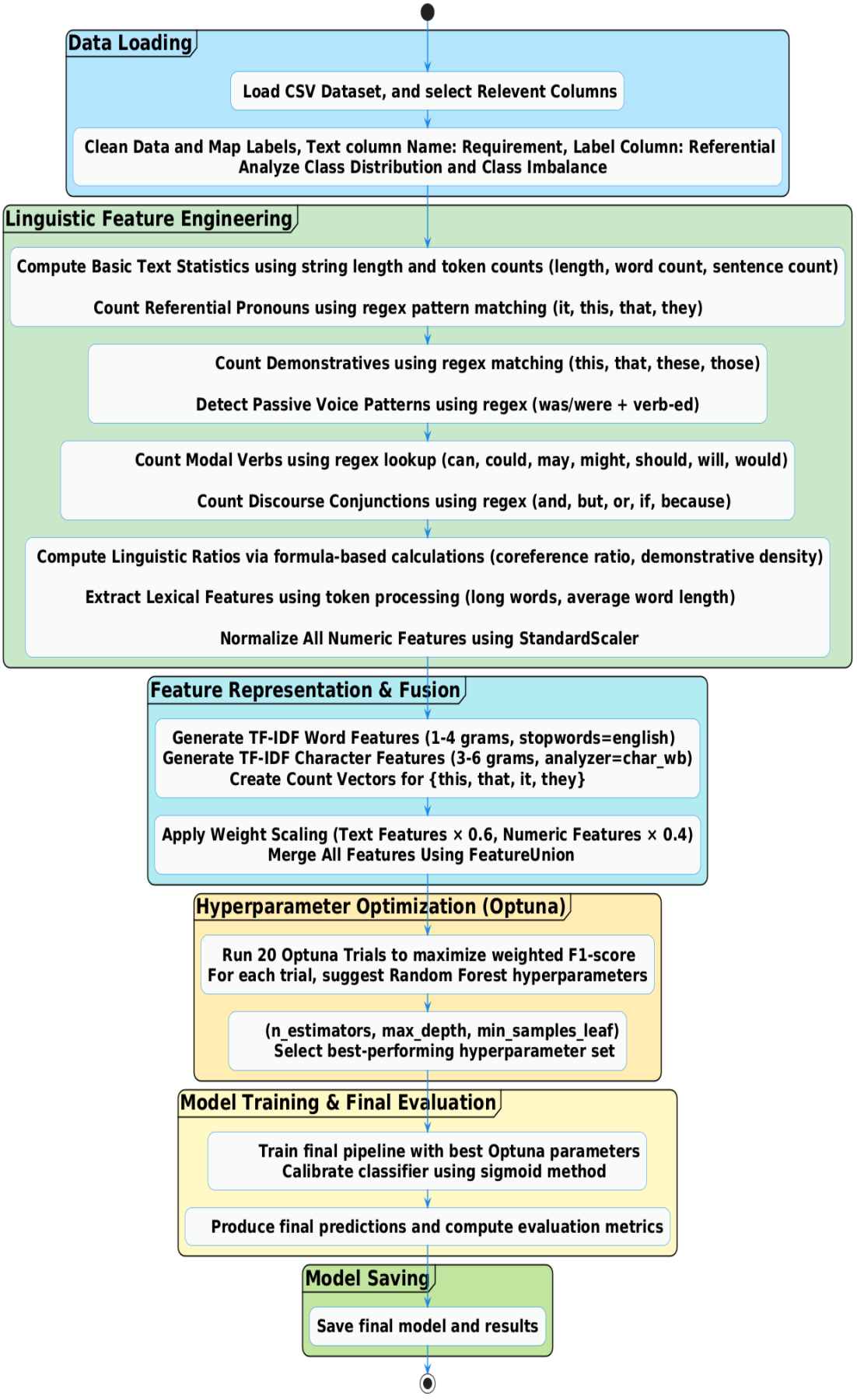


FIGURE 3.2: Referential Ambiguity Detection Flow Diagram

columns due to possible variances in labeling these columns as text or label. All textual data under the 'Requirement column was rendered into the standard string and remove extraneous whitespace for uniformity and neatness. The labels in the column of labels under referential were converted to integers to be used directly, without modification, in the modelling classification. The missing textual entries and labels were filtered out to retain the data integrity.

3.2.2 Advanced Feature Engineering

The system employs a custom feature extraction component (EnhancedFeatureExtractor) that transforms each requirement into a structured numerical vector. The extractor uses vectorized operations and regular-expression matching to identify linguistic patterns known to contribute to referential ambiguity. The engineered features are grouped into six categories, described below. Structural Features quantify the basic structural properties of a requirement. Character length is computed as the total number of characters in the text. Word count is obtained by splitting the text on whitespace, and sentence count is approximated by identifying terminal punctuation marks such as ".", "!" and "?". These indicators help assess the overall complexity of the statement prior to deeper linguistic analysis.

Because referential ambiguity frequently arises from unclear antecedents, several features quantify the presence of pronouns and demonstratives. Pronoun frequency counts occurrences of it, this, that, they, while demonstrative frequency captures this, that, these, those. Possessive constructions (e.g., "system's", "of the user") are detected to highlight relational expressions that may obscure the intended referent. Together, these features measure how heavily a requirement relies on potentially ambiguous references. Multiple grammatical constructions associated with unclear agency or implicit referencing are identified. Passive voice is detected by matching auxiliary verbs such as was or were followed by a verb in past participle form. Clause-initial markers such as which is or that is are captured to reflect embedded descriptive clauses that may obscure the referent (corresponding to the that_deletion feature). Modal verb frequency (e.g., can, may, should, would) is

recorded because modals weaken determinism, while conjunction frequency (e.g., and, but, if, because) identifies multi-clausal structures. These features collectively capture syntactic patterns that contribute to referential ambiguity.

Two binary features measure the presence of punctuation that deviates from standard requirement writing conventions. A question indicator is set when the requirement contains "?", and an exclamation indicator when "!" is present. These indicators are binary (1 if present, 0 otherwise) and help flag informal or structurally atypical expressions.

Two ratio-based features quantify the concentration of referential markers within the statement. Coreference ratio divides the number of pronouns by the total word count, while demonstrative density divides the number of demonstratives by the number of sentences. A small constant ensures numerical stability for short statements. These ratios normalize referential indicators relative to statement length and structure, providing a more interpretable measure of referential intensity.

Additional features capture higher-level stylistic and ambiguity-related tendencies. A long-sentence flag activates when a requirement exceeds 25 words, indicating potential cognitive load. Ambiguous use of "this" is detected when it appears without an immediately following verb, suggesting an unstated referent. The count of long words measures tokens exceeding six characters, and average word length is computed as the mean character length of all tokens. These features capture lexical and structural complexity linked to ambiguity.

All engineered features are combined into a dense numerical matrix and standardized using z-score normalization. This ensures that count-based, ratio-based, and binary features contribute proportionately to the classifier and prevents high-magnitude features from dominating the learned decision boundaries.

3.2.3 Feature Representation and Pipeline Construction

The proposed system integrates two distinct types of feature representations high-dimensional textual vectors and dense engineered linguistic features into a unified

processing pipeline. This is achieved through a parallel, multi-branch architecture, ensuring that each feature type contributes appropriately to the final classification.

Textual information is encoded using three complementary vectorization components. Each component operates independently, and all outputs are merged through a parallel combination mechanism.

i. Word-Level TF–IDF Features: A term frequency - inverse document frequency (TF–IDF) vectorizer converts each requirement into a weighted representation of unigrams to four-grams. It uses a maximum of 8,000 features, with English stopwords removed using scikit-learn's default TF–IDF weighting scheme. This captures lexical content, short phrasing patterns, and contextual fragments relevant to referential expressions.

ii. Character-Level TF–IDF Features: A second TF–IDF model extracts three- to six-character n-grams using a word-boundary-sensitive analyzer. It uses a maximum of 4,000 subword features. This representation detects morphological patterns, stems, affixes, and partial tokens that may indicate referential usage or structural irregularities.

iii. Binary Pronoun-Presence Features: A compact CountVectorizer encodes the presence of key pronouns (this, that, it, they) using a fixed vocabulary. This module produces a sparse binary indicator vector highlighting whether each pronoun appears in the requirement.

All three textual representations are combined using a FeatureUnion, which executes them in parallel and concatenates their outputs into a unified sparse matrix. This preserves the independence of each text-based feature space. The engineered linguistic features produced by the EnhancedFeatureExtractor form a dense numeric feature set. These features capture structural complexity, referential density, grammatical constructions, and ambiguity patterns. Before integration, the numeric features undergo z-score standardization, ensuring uniform scaling. This prevents high-magnitude count features from overwhelming ratio-based or binary features during model training. To balance the influence of the high-dimensional textual features and the lower-dimensional linguistic features, explicit weighting

is applied to each branch. Textual features are scaled by 0.60, emphasizing lexical and subword patterns. Numeric linguistic features are scaled by 0.40, ensuring handcrafted ambiguity cues remain influential without dominating the representation. The weighting is applied through lightweight transformation components, and the outputs of both branches are merged again using FeatureUnion into a single composite feature matrix. This unified representation forms the input to the classifier.

TABLE 3.1: Performance of Different Text and Numeric Feature Weights for Referential Ambiguity Detection

Configuration	Metric	Class 0	Class 1	Macro Avg	Weighted Avg	Accuracy
Text = 0.6 Numeric = 0.4	Precision	0.87	1.00	0.93	0.93	0.92
	Recall	1.00	0.85	0.92	0.92	
	F1-score	0.93	0.92	0.92	0.92	
Text = 0.7 Numeric = 0.3	Precision	0.87	1.00	0.93	0.93	0.92
	Recall	1.00	0.85	0.92	0.92	
	F1-score	0.93	0.92	0.92	0.92	
Text = 0.5 Numeric = 0.5	Precision	0.81	1.00	0.91	0.91	0.88
	Recall	1.00	0.77	0.88	0.89	
	F1-score	0.90	0.87	0.88	0.88	
Text = 0.4 Numeric = 0.6	Precision	0.86	0.92	0.89	0.89	0.89
	Recall	0.92	0.85	0.89	0.89	
	F1-score	0.89	0.88	0.88	0.88	
Text = 0.3 Numeric = 0.7	Precision	0.86	0.92	0.89	0.89	0.89
	Recall	0.92	0.85	0.89	0.89	
	F1-score	0.89	0.88	0.88	0.88	

The architecture uses a nested FeatureUnion design: the textual branch is itself a FeatureUnion of three parallel vectorizers, and a second FeatureUnion merges the weighted textual branch with the weighted numeric branch into a final composite feature matrix. To determine an appropriate weighting strategy for combining textual TF–IDF features with engineered linguistic features, five weighting configurations were evaluated. The weighted F1-score was selected as the primary metric because it provides a balanced measure of performance for both classes and is more reliable than accuracy for smell detection. Across the experiments,

the 0.6/0.4 and 0.7/0.3 configurations yielded the highest weighted F1-score (0.92). However, the 0.6/0.4 configuration demonstrated more consistent results, showing a stable balance between precision and recall for both classes.

The remaining configurations 0.5/0.5, 0.4/0.6, and 0.3/0.7 consistently produced lower weighted F1-scores (0.88), indicating reduced effectiveness when the contribution of textual features is decreased. Since referential ambiguity is largely driven by lexical patterns such as pronouns and demonstratives, the results confirm that a higher contribution from textual features improves detection performance while still allowing engineered features to play a meaningful complementary role. Based on these empirical findings, the 0.6/0.4 weighting scheme was selected as the final configuration because it provides the most stable and highest weighted F1-score across all evaluated settings as shown in table 3.1.

3.2.4 Classification with Random Forest

The combined feature representation is processed by a Random Forest classifier, selected for its robustness to noise, ability to handle heterogeneous feature types, and suitability for medium-sized datasets. Random Forests operate by training multiple decision trees on random subsets of both samples and features, and combining their predictions through majority voting. This ensemble behaviour reduces overfitting and captures non-linear dependencies between lexical patterns and engineered linguistic indicators. Unlike fixed-parameter configurations, the Random Forest model in this study is tuned automatically using Optuna.

During the optimization process, Optuna explores a search space spanning 200 – 800 trees (n_estimators), maximum depths between 8 and 20 (max_depth), and minimum leaf sizes between 1 and 5 (min_samples_leaf). Each sampled configuration is evaluated using the weighted F1-score on a validation split, and the configuration that yields the highest score is selected. This approach ensures that the classifier is adapted specifically to the linguistic and statistical characteristics of the referential ambiguity dataset. The classifier uses class_weight='balanced_subsample' to mitigate class imbalance by recalculating class weights individually for each

tree during bootstrap sampling. All models are trained with parallel computation (n_jobs=-1) to exploit available CPU cores. The balanced-subsample strategy ensures that minority ambiguous examples influence tree construction proportionately, without modifying the underlying dataset through over sampling or under sampling. This results in improved sensitivity to ambiguous statements while maintaining generalization.

3.2.5 Model Training and Evaluation

After constructing the feature architecture and defining the Random Forest classifier, the model is trained using the training portion of the dataset. A stratified train test split is applied to preserve the original class distribution of referential and non-referential requirements. Model training incorporates hyperparameter optimization through Optuna, which evaluates multiple candidate configurations of the Random Forest (e.g., n_estimators, max_depth, min_samples_leaf) and selects the combination that maximizes the weighted F1-score on the validation split. The final model is then re-trained end-to-end on the full training set using the optimal configuration identified during tuning.

Because the entire pipeline is implemented using scikit-learn's Pipeline and FeatureUnion constructs, all preprocessing steps including TF–IDF vectorization, binary pronoun-presence encoding, linguistic feature extraction, and z-score normalization are applied consistently to both the training and test data without risk of leakage. The evaluation is conducted on the held-out test set using standard performance metrics, including precision, recall, F1-score, and the confusion matrix.

These metrics provide a comprehensive assessment of the model's effectiveness in detecting referential ambiguity. The calibrated and threshold-optimized predictions (described in Section 3.2.6) are incorporated to ensure that the final reported performance reflects the most reliable decision boundary produced by the model.

3.2.6 Probability Calibration and Threshold Optimization

Because Random Forests do not naturally produce well calibrated probability estimates, a probability calibration stage is applied to improve the reliability of the

predicted class probabilities. The trained classifier is wrapped using sigmoid-based Platt scaling via the CalibratedClassifierCV framework operating in prefit mode.

This procedure transforms the raw model outputs into calibrated probability estimates that more accurately reflect the true likelihood of a requirement belonging to the positive class.

Calibration enhances interpretability and provides a more stable basis for threshold selection. After calibration, a continuous threshold search is performed to identify the most effective probability to label conversion point. A Precision Recall curve is constructed, and the F1-score is computed at each candidate threshold.

The threshold that yields the maximum F1-score is selected as the optimal decision boundary. This optimized threshold is incorporated directly into the final prediction process, ensuring that the classification results are produced using the most effective decision criterion particularly beneficial in detecting requirement smells where class imbalance or ambiguous linguistic patterns may render the default 0.5 threshold suboptimal.

3.3 Proposed Methodology for Unverifiability Detection

Unverifiability in requirements engineering (RE) is defined as requirements that cannot be tested, measured, and concretely defined that may be the cause of either under-understanding or under-implementation through software change development. In order to overcome this problem, a systematic unverifiability detection methodology has been devised that categorizes requirements as verifiable or unverifiable through a combination of linguistic/structure feature extraction and machine learning. This chapter precisely explains methodology, tool, data preprocessing procedure, feature representation, classification pipeline, evaluation processes which were used in the course of unverifiability detection process and contains a full framework against which to detect unverifiable statements in natural language requirements. Detailed flow diagram is shown in figure 3.3.

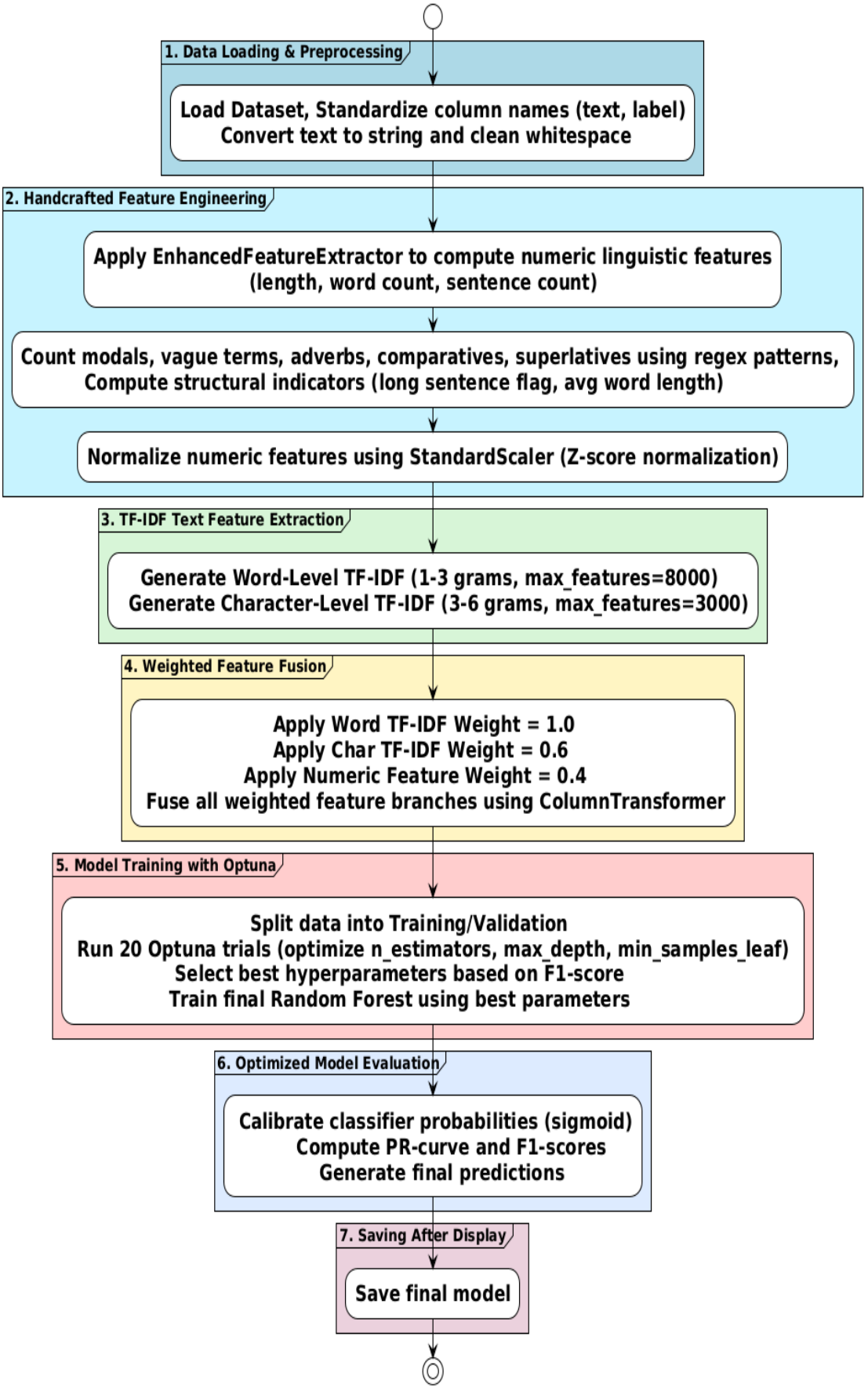


FIGURE 3.3: Unverifiability Detection Flow Diagram

3.3.1 Data Preparation and Preprocessing

The research is based on the dataset that includes the requirement statements with labelings that denote their status in terms of unverifiability. The original data is constituted with columns with different names in different sources. In order to establish standardization, the column which will contain the requirement texts will be named uniformly as "Requirement" with the binary column of labeling requirement statements as unverifiable (1) or verifiable (0) being renamed as "Unverifiable".

To facilitate the identification of these columns in the face of possible inconsistencies in terms of naming, the methodology incorporates a loose metric to accommodate these columns by searching keywords such as textual data often termed as requirement or text and the classification labels are searched with keywords such as unverifiability, label or target. This consistency in referencing assurances the process of preprocessing and modeling are consistent.

On loading, all the textual data types in the column headed Requirement are normalized into a consistent string type and stripped of any extraneous whitespace to ensure higher data quality. Missing text entries are dropped to preserve data integrity and any missing value in labels is imputed by using the most frequent label class to avoid bias introduced by incomplete labels.

The class distribution will also be reported when loading data to point out any possible imbalance between verifiable and non-verifiable classes that may influence the training and evaluation of the models. This resulting clean and standardized data is used as the baseline of feature calculation and modeling.

3.3.2 Advanced Feature Engineering

Linguistic features specifics are obtained in the form of a custom transformer that detects specific linguistic aspects of each used textual input. This involves using regular expression matching to count sentences based on word patterns and word categories-i.e. modal verbs (can, must), non specific words (some, various), and

sentence ending punctuation (periods, exclamation points, question mark). Other attributes are based on text tokenization and string operations: they include the calculation of the total character count, the number of words counted as separated by whitespace delimiters, and average length of a word (mean size of the tokens). The combination of all these features forms the linguistic characteristics that constitute signs of unverifiability and become the additional feed of data along with a textual one to be classified. A set of relevant features that can meaningfully capture the linguistic indications related to unverifiability is extracted with respect to the textual information. These features quantify the basic structure of the requirement: character length of the requirement text (str.len()); word count, computed by splitting tokens (str.split().str.len()); and sentence count, approximated using the number of terminal punctuation marks (str.count(r'[.!?]')). These indicators highlight excessively long or complex requirement statements, which often correlate with unverifiability. Unverifiable requirements frequently contain vague or subjective expressions. The extractor identifies modal verbs such as may, might, could, should, often indicating non-verifiable conditions; vague terms, including some, various, many, appropriate, user-friendly, representing underspecified criteria; and subjective adverbs such as quickly, easily, significantly, indicating unverifiable performance descriptions. Patterns are matched using regular expressions applied to the requirement text. Comparative forms (e.g., better, faster, improved) and superlatives (e.g., best, fastest, most) are extracted because they describe relative performance without quantifiable baselines. Comparatives detected using: '(better, faster, cheaper, improved)' and superlatives detected using: '(best, fastest, most, least)'. These forms frequently contribute to unverifiability due to their inherently subjective interpretations.

Additional indicators reflect structural complexity and potential ambiguity: long sentence flag, enabled if the requirement exceeds 25 words; and average word length, capturing lexical density and readability. These metrics help detect specifications that mix multiple unverifiable statements or describe vague high-level expectations.

All handcrafted features are combined into a dense numerical matrix. Before fusion

with textual vectors, the feature matrix is standardized using z-score normalization (StandardScaler) to ensure consistent scaling across count-based, ratio-based, and binary signals.

3.3.3 Feature Representation and Pipeline Construction

The unverifiability detection pipeline integrates two types of feature representations: sparse textual features extracted using TF–IDF vectorization and dense handcrafted linguistic features engineered to capture unverifiable patterns in natural language requirements. These feature sets are combined using a weighted multi-branch architecture, ensuring that both lexical patterns and linguistic cues jointly contribute to classification. Numeric Linguistic Features: Beyond the TF–IDF representations, the pipeline includes a numeric feature branch that encodes sentence-level and lexical cues strongly associated with unverifiable expressions.

These handcrafted indicators capture properties that TF–IDF alone may overlook, such as requirement length, total word count, number of sentences, frequency of modal verbs (e.g., should, may), vague terms (e.g., some, various), subjective adverbs (e.g., quickly, easily), comparative and superlative forms, long-sentence flags, and average word length. All numeric features are standardized using a StandardScaler and assigned a weight of 0.4 within the multi-branch fusion, ensuring balanced contribution alongside the textual TF–IDF representations.

Textual TF–IDF Encoding: Two TF–IDF vectorizers operate in parallel within the pipeline one at the word level and the other at the character level. Both vectorizers run automatically during model training and inference. Word-Level TF–IDF Encoding: This vectorizer converts each requirement into a sparse numerical vector based on the relative importance of its words and short phrases. The transformation occurs through the following steps:

Step 1: Tokenization: The input text (inside the “text” column) is split into tokens using whitespace and punctuation boundaries. Each token represents a word.

Step 2: N-gram Generation: The vectorizer constructs n-grams for n = 1, 2, and 3. For example, the text: “The system should respond quickly” produces

unigrams: the, system, should, respond, quickly; bigrams: the system, system should, should respond, respond quickly; and trigrams: the system should, system should respond, should respond quickly.

Step 3: Vocabulary Construction: During training, the vectorizer identifies up to 8,000 of the most frequent n-grams across the dataset after removing English stopwords.

Step 4: TF–IDF Weight Calculation: For every requirement, Term Frequency (TF): how often each n-gram appears in the requirement. Sublinear scaling transforms TF as "TF" = 1 + log("term count"). Inverse Document Frequency (IDF): how rare each n-gram is across the entire dataset: "IDF" = log(N / (1 + df)), where N is the total number of documents and df is the number of documents containing the n-gram.

Step 5: Sparse Matrix Output: Each requirement is encoded as a high-dimensional sparse vector (up to 8,000 columns). Most values are zero because each requirement contains only a small subset of all possible n-grams. This representation captures vague adjectives, modal patterns, and loosely defined requirement expressions because such phrases influence TF–IDF values directly. Character-Level TF–IDF Encoding: The character n-gram vectorizer processes each requirement at the subword level, enabling the detection of morphological and structural patterns associated with unverifiable phrasing.

Step 1: Character Sequence Extraction: Each requirement is broken into overlapping sequences of 3 to 6 characters. Example for the word "quickly": 3-grams: qui, uic, ick, ckl, kly; 4-grams: quic, uick, ickl, ckly; and up to 6-grams.

Step 2: Word-Boundary Analyzer (char_wb): The analyzer extracts character sequences within word boundaries, avoiding cross-word artifacts. For example, in "fast execution", character n-grams are extracted separately from fast and execution, but not across the space between them.

Step 3: Vocabulary Construction: During training, up to 3,000 of the most informative character n-grams are retained.

Step 4: TF–IDF Weighting: As with the word-level vectorizer, TF–IDF weights are computed for each character pattern.

Step 5: Sparse Matrix Output: Each requirement is encoded as a sparse vector of up to 3,000 character-level features.

This representation captures vague morphological patterns, suffixes, and stylistic indicators often present in unverifiable requirements (e.g., -ly, -ish, -able, improved, optimized, etc.).

3.3.4 Pipeline Implementation and Feature Fusion

The unverifiability detection pipeline is implemented as a weighted multi-branch architecture that fuses sparse textual encodings with dense handcrafted linguistic indicators. Textual encodings are produced by two TF–IDF branches: a word-level TF–IDF vectorizer (max_features = 8000, ngram_range = (1,3), stop_words = 'english', sublinear_tf = True) and a character-level TF–IDF vectorizer (max_features = 3000, ngram_range = (3,6), analyzer = 'char_wb'). Handcrafted numerical features such as requirement length, word count, sentence count, modal verbs, vague terms, subjective adverbs, comparative and superlative forms, long-sentence flags, and average word length are extracted by the EnhancedFeatureExtractor and standardized using a StandardScaler. Feature weighting is applied through dedicated WeightApplier modules: the word-level branch receives full weight (1.0), the character level branch is down-weighted (0.6), and the numeric feature branch receives a weight of 0.4. The fusion is implemented using a ColumnTransformer to ensure consistent, modular, and fully integrated end-to-end transformation during training and inference. In the unverifiability detector, three separate feature weights were required because the feature space consists of three heterogeneous and statistically distinct components: (i) word-level TF–IDF, which captures semantic vagueness, (ii) character-level TF–IDF, which captures stylistic and morphological cues, and (iii) handcrafted linguistic features, which capture rule-based indicators of unverifiability. Merging word and character TF–IDF into a single "text" block as done in the referential ambiguity model would eliminate the ability to

control their relative influence and may allow character level noise to dominate word-level semantic cues. Therefore, three independent weights provide more balanced and effective feature fusion for unverifiability detection. Word-level TF–IDF and character-level TF–IDF were kept as separate feature branches because they capture different linguistic phenomena relevant to unverifiability.

Word TF–IDF models semantic vagueness and therefore requires full weight,whereas character TF–IDF captures morphological cues and is inherently noisier, requiring a reduced weight.

Combining both branches into a single representation would enforce identical weighting despite their different predictive strengths, imbalance the feature space due to dimensionality differences, and reduce interpretability. Accordingly, the pipeline preserves them as distinct, independently weighted components.

3.3.5 Classifier Configuration and Training Procedure

The fused feature vector is passed to a Random Forest ensemble designed to capture non-linear interactions between TF–IDF representations and engineered linguistic indicators. Unlike fixed parameter models, the Random Forest configuration for unverifiability detection is optimized automatically using Optuna. During the optimization process, key hyperparameters such as the number of trees (n_estimators), maximum depth (max_depth), and minimum samples per leaf (min_samples_leaf) are sampled within predefined ranges and evaluated using the weighted F1-score on the validation set. Specifically, Optuna explores values in the following ranges:

- *n_estimators*: 200–700
- *max_depth*: 8–25
- *min_samples_leaf*: 1–5

The Random Forest uses class_weight='balanced_subsample' to dynamically adjust class importance during bootstrap sampling, mitigating the strong class imbalance between verifiable and unverifiable requirements.

All models are trained with n_jobs=-1 to utilize all CPU cores. After Optuna identifies the best performing hyperparameter configuration, a final Random Forest model is trained end to end on the training split. The training follows a stratified 75/25 or 80/20 split (depending on dataset size) to preserve the original class distribution.

Because the complete pipeline including TF–IDF vectorizers, feature scalers, and the classifier is encapsulated in a single scikit-learn Pipeline, all transformations are fitted consistently within the same data fold, ensuring reproducibility and preventing data leakage.

3.3.6 Probability Calibration and Threshold Optimization

Raw probability estimates produced by ensemble classifiers such as Random Forests can often be miscalibrated, meaning that the predicted probabilities do not accurately reflect the true likelihood of a requirement being unverifiable. To address this, the methodology applies probability calibration using CalibratedClassifierCV with a sigmoid (Platt scaling) transformation and cv='prefit'.

The calibration model is fitted on the held-out validation split, ensuring that the probability adjustments do not leak information from the training set. Once calibrated probabilities are obtained for each requirement, the system evaluates multiple candidate decision thresholds using the precision recall curve.

For every possible threshold, precision, recall, and the corresponding sample-level F1-score are computed. The threshold that maximizes the F1-score is selected as the optimal operating point, allowing the classifier to balance sensitivity toward minority unverifiable requirements with overall predictive reliability.

This process replaces the default 0.5 threshold, which is often suboptimal in the presence of class imbalance or ambiguous linguistic patterns. The optimized threshold is then applied to convert calibrated probabilities into final binary predictions. The evaluation includes the full range of standard classification metrics: per-class precision, recall, and F1-score; weighted-average performance summaries;

and confusion matrices computed for both the default and optimized-threshold predictions.

Additionally, a detailed prediction summary containing the requirement text, actual label, predicted label, and calibrated probability is exported to CSV format to support downstream error analysis and interpretability studies.

3.3.7 Classification with Random Forest

Unverifiability is classified with the help of a Random Forest machine-learning algorithm, which is an ensemble of decision trees. The trees are also trained on a stochastic sub-sample of the training data, and a stochastic sub-sample of features, techniques known as bagging and feature randomness respectively. Within the classification process, all of the trees produce separate class predictions; final prediction is reached by majority voting across the ensemble. Such approach reduces the overfitting risk of single decision trees and improves the generalization. In the present study, the Random Forest classifier is not configured with fixed hyperparameters.

Instead, its key parameters including the number of trees (n_estimators), maximum tree depth (max_depth), and minimum samples per leaf (min_samples_leaf) are optimized automatically using Optuna. During this process, the search space spans 200–700 trees, depths between 8 and 25, and leaf sizes between 1 and 5. The classifier uses class_weight='balanced_subsample' to mitigate the strong class imbalance in unverifiable requirements by adjusting class importance dynamically in each bootstrap sample. This configuration improves sensitivity to the minority class while maintaining robust generalization.

3.3.8 Model Training and Evaluation

The Random Forest classifier is trained on the processed dataset using the multi-branch feature representation that integrates both TF–IDF encodings and engineered numerical indicators. To ensure a reliable evaluation, the dataset is divided into training and validation subsets using stratified sampling, thereby preserving

the original class distribution of verifiable and unverifiable requirements. Unlike traditional configurations with fixed hyperparameters, the present study employs Optuna to automatically optimize key Random Forest parameters. During training, Optuna evaluates multiple candidate configurations each defined by different values of n_estimators, max_depth, and min_samples_leaf and selects the combination that maximizes the weighted F1-score on the validation set. After the optimal hyperparameters are identified, a final model is trained end-to-end on the training split. Beyond discrete class labels, the model also generates calibrated probability estimates for each requirement. These probabilities are produced using a sigmoid-based CalibratedClassifierCV fitted on the validation split. The calibrated scores enable threshold-based decision making: by computing precision–recall curves and corresponding F1-scores, an optimal decision threshold is selected to balance sensitivity toward unverifiable requirements and overall predictive performance. This evaluation strategy provides a robust and unbiased assessment of the model's ability to recognize unverifiable statements in natural language requirements.

3.4 Proposed Methodology for Subjectivity Detection

Subjectivity in requirements engineering (RE) cannot be precisely defined, but roughly it means an imprecise, opinion-based, or ambiguous language that might cause the misinterpretation of the software development. In order to cope with this issue, a greater subjectivity detection methodology was created in order to categorize requirements as subjective or objective by use of both rule-based and machine learning approaches. This part describes the methodology, tools, data set, and data processing required and the evaluation method adopted throughout the subjectivity detection procedure, and concentrates on the data set that is exploited by this study. Complete detection process is shown in figure 3.4.

3.4.1 Data Preparation and Preprocessing

The study is grounded on a collection of natural language software requirements with an annotation on the subjectivity data. The requirements are presented in a

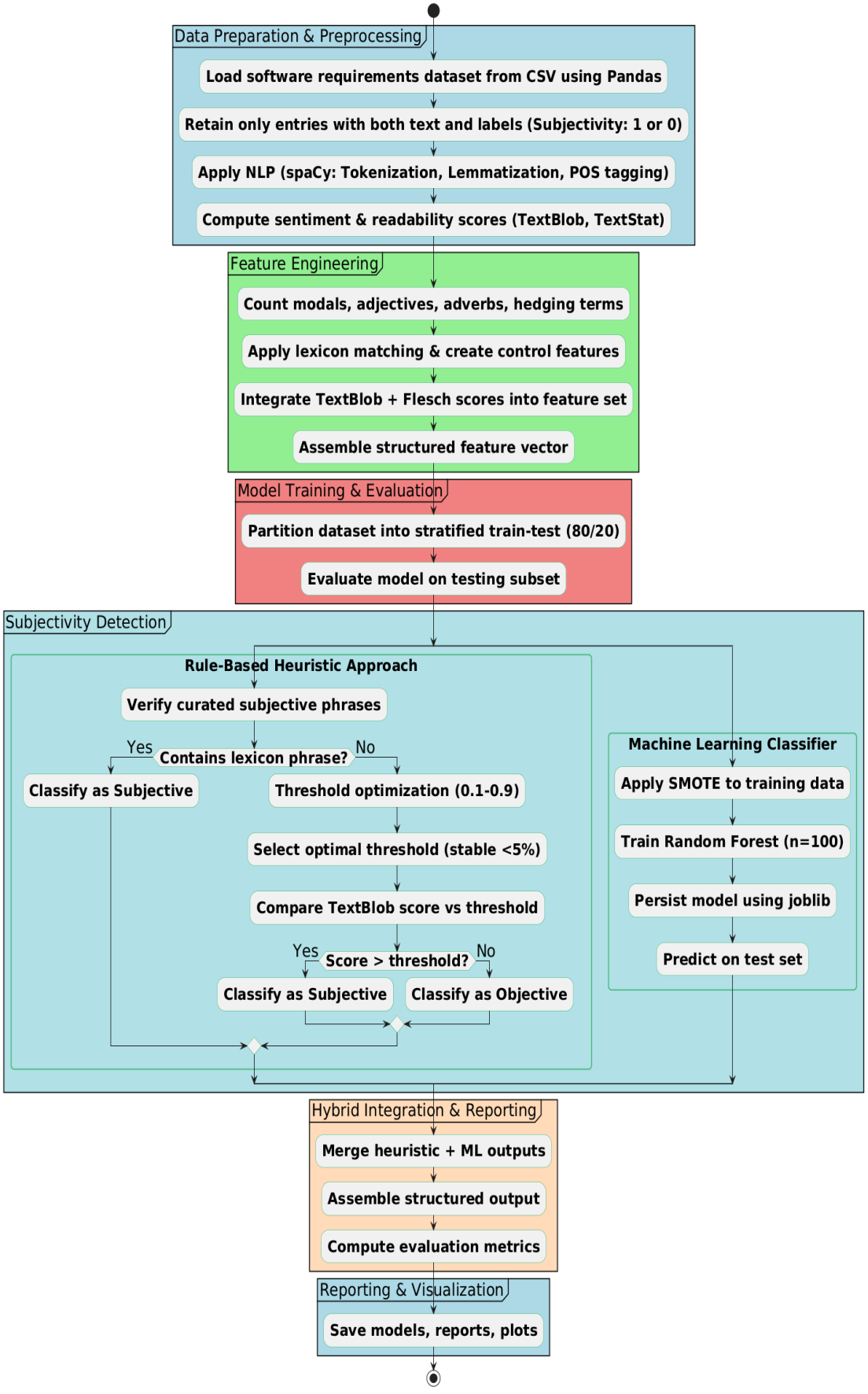


FIGURE 3.4: Subjectivity Detection Flow Diagram

tabular dataset as a row with two key attributes: (i) Requirement column where the textual statements in natural language are included, (ii) Subjectivity column where binary labels are included, where 1 means that the requirement is subjective and 0 means that it is objective. Only rows having both requirement text and matching corresponding subjectivity labels are stored so that, the training and evaluation dataset is clean, consistent and fully annotated. Natural language processing is applied to each requirement statement to derive linguistic and semantic information that can be used to analyze subjectivity. SpaCy en_core_web is used to tokenize, lemmatize and part of speech (pos) tag, these allow recognizing grammatical elements within a sentence, including modal verbs, adjectives and adverbs. Subjectivity and polarity scores are calculated with help of the TextBlob library and allow obtaining signals associated with sentiment. Moreover, the Flesch Reading Ease score (an indicator of readability) is computed with TextStat. By these preprocessing activities, those raw requirement statements are turned into linguistically rich representations, which are both compatible with rule-based heuristic models as well as with trained supervised learning models.

3.4.2 Feature Engineering

Requirement statements are translated into structured feature vectors in order to train machine learning classifiers and to aid the rule-based reasoning. These characteristics denote linguistic, semantic indicators related to expressions of subjectivity. Features that are engineered correspond to:

i. Modal verb count: May, might, could, and should are used with tokens as these are commonly used to mean uncertainty, or non-promising.

ii. Hedging terms: Perhaps, maybe, possibly and likely are words that are identified using curated lexicon.

iii. Adjectives and adverbs: Due to the presence of descriptive and evaluative expressions, extracted out of POS tags (ADJ and ADV).

iv. -ly adverbs: The adverbs with an -ly ending: -ly adverbs (easy, safely, etc.) are also registered, since they often attach herself to the subject of describing.

v. Subjective phrases: Phrase matching is performed on a curated list of domain specific subjective expressions (User friendly, easy to use, as needed, etc). This lexicon was built by merging words of the previous literature and manually discovered phrases of actual world needs.

vi. Readability score: Flesch Reading Ease value is added only to include the complexity of the statement. Word count: This is the number of words embedded in the requirement as a control element.

All characteristics are indexed as organized dictionaries and converted into numerical vectors, which can be trained and assessed. All features are normalized to the same scale in order to compare features during a model training phase. This process helps avoid influencing the classifier by features that have higher absolute values (e.g. word counts).

3.4.3 Rule Based Detection

The rule based detection tool is based on the explicitly defined linguistic heuristics to differentiate subjective requirements and objective requirements. For categorisation to be transparent and interpretable, this layer uses sentiment-based thresholding and direct phrase matching.

It starts with phrase matching that utilizes an edited word bag of biased phrases. And a requirement which contains one of these phrases is immediately Subjective. The vocabulary contains both generic terms (e.g., convenient, ideally, secure) and terms that were important in the RE domain (e.g., automatic, efficient), revealed after studying the data. With the claim explicitly pegged to these written phrases, the resulting methodology makes the results of classification traceable and linguistically-motivated.

In case a subjective phrase is not identified, a subjectivity threshold parameter is obtained based on the TextBlob subjectivity score. Although the traditional benchmark of 0.5 is used in the TextBlob, that is not an arbitrary number. Rather, a set of threshold values (between 0.1 and 0.9 by intervals of 0.05) are exhaustively

tested on the labelled dataset. The value with the most consistent and reliable performance in the sense of its F1-score is chosen. Through this, the threshold involved becomes empirically based, not theoretically based. Also, the threshold chosen has to show consistent fidelity to the various training and testing splits of the dataset. This is in practice equivalent to ensure that variation in the accuracy or F1-score is less than a reasonable margin (e.g., less than 5 percent). Because stability is a criterion enforced by the methodology, it does not overfit to a given data partition and produces results that are generalizable.

In conclusion, the rule-based subjectivity detection algorithm is a combination of thresholding and direct phrase matching, because the detectors find optimized empirically has been developed using constrained experiments. The design offers clearness and straightforwardness and is likewise strong and adaptable. Since the lexicon and thresholds are both documented explicitly, the rules may be revised or extended in the future, thus, the approach is sustainable and can be applied in many requirements engineering situations.

Following are the rules for subjectivity detection in Extended Backus-Naur Form (EBNF), a formal notation used to describe language syntax.

```
(* --- Basic Lexical Elements --- *)

LETTER = "A" | "B" | ... | "Z" | "a" | "b" | ... | "z" ;
DIGIT = "0" | "1" | ... | "9" ;
WHITESPACE = " " | "\t" | "\n" | "\r" ;

(* --- Terminals specific to Subjectivity Detection --- *)

SUBJECTIVE_PHRASE = "user friendly"
                  | "easy to use" | "should be able to"
                  | "as needed"   | "when necessary"
                  | "if possible" | "may"       | "might"
                  | "could"       | "would"     | "preferably"
                  | "if appropriate"            | "as required"
                  | "when needed" | "safely"    | "easy"
```

```
                    | "easily"      | "valid"     | "automatic"
                    | "successful"  | "external"  | "periodic"
                    | "sensitive"   | "another"   | "part"
                    | "case"        | "unit"      | "default"
                    | "effective" ;

SUBJECTIVITY_SCORE = DIGIT , '.' , DIGIT+ ;
THRESHOLD = "0.5" | SUBJECTIVITY_SCORE ;
CLASSIFICATION = "Subjective" | "Objective" | "Invalid" ;
SCORE = "0.0" | "1.0" | SUBJECTIVITY_SCORE ;

(* --- Non-terminals and Rules --- *)

TEXT = (LETTER | DIGIT | WHITESPACE | PUNCTUATION)+ ;

SUBJECTIVITY_DETECTION = INPUT_VALIDATION
                       | PHRASE_MATCHING
                       | THRESHOLD_CLASSIFICATION ;

(* Rule 1: Input Validation *)
INPUT_VALIDATION = EMPTY_INPUT
                 | NON_STRING_INPUT ;

EMPTY_INPUT = "" ; (* Returns ("Invalid", "0.0") *)
NON_STRING_INPUT = (* non-string input *) ; (* Returns ("Invalid",
    "0.0") *)

(* Rule 2: Phrase Matching *)
(* A text is classified as Subjective if it contains any of the
    defined subjective phrases. *)
PHRASE_MATCHING = TEXT , (SUBJECTIVE_PHRASE)+ ; (* Returns
    ("Subjective", "1.0") *)

(* Rule 3: Threshold-based Classification *)
```

```
(* A text is classified based on a model-generated score if no
   explicit subjective phrase is found. *)
THRESHOLD_CLASSIFICATION = TEXT , (!SUBJECTIVE_PHRASE) ,
   (ASSIGN_SCORE) ;

ASSIGN_SCORE = (* function that returns a SUBJECTIVITY_SCORE based
   on a model *) ;
(* The final classification is determined by the score relative to
   the threshold: *)
(* If SUBJECTIVITY_SCORE >= THRESHOLD then returns ("Subjective",
   SUBJECTIVITY_SCORE) *)
(* Else if SUBJECTIVITY_SCORE < THRESHOLD then returns
   ("Objective", SUBJECTIVITY_SCORE) *)
```

Listing 3.1: EBNF Rules for Subjectivity Detection

3.4.4 Machine Learning Classifier

The machine learning classifier enhances this rule-based work by applying statistical learning in recognizing a pattern in subjective requirements that might not have been represented with explicit language rules. This part is presented to acquire the discriminatory characteristics of labelled material and provide it with an empirical basis to recognise subjectivity. The process started with data partitioning, in data partitioning a dataset is divided into a training set and testing set through stratified sampling. The stratification also ensures that the subjective and objective needs of the two groups are balanced in terms of the distribution of classes to ensure representativeness. The methodology solves this problem because subjective requirements additionally tend to be fewer than objective ones, therefore implementing the Synthetic Minority Over-sampling Technique (SMOTE). The step creates artificial examples of the minority class (subjective requirements) in the training set, so that the classifier sees an equal amount of exposure to both classes during learning. In such a manner, the model will not be biased towards the majority class, and it will detect the appearance of the minority-class better.

A Random Forest algorithm of classification is the basic learning algorithm; it is used to balance three criteria: predictive accuracy, speed and memory to run the algorithms and finally, the number of decision trees utilized (100). Every tree gets trained on a random sample of features and data points and predictions are combined by majority voting. This serves the purpose of improving the classification results and reducing the degree of over hyphenation. The trained classifier is stored with joblib library after training and can be reused and reproduced in the future without executing the computations there again. The forecasts are then made on nonobserved test data, and model generalization is evaluated independent of the predictions. The machine learning classifier provides credible performance in the detection of subjective requirements by utilizing previously mentioned combination of balanced data preprocessing and powerful ensemble learning method.

3.4.5 Model Training and Evaluation

The functioning of the rule-based and machine learning component is evaluated in a systematic manner aligned with evaluation protocol. The dataset is stratified sampled to create training and testing groups in order to maintain the same distribution of classes. Furthermore, k-fold cross-validation is implemented since it removes the chance of the results of the cross-validation being biased against a given data set and gives the model a closer global prediction. In the case of the rule-based system, the subjectivity threshold is fixed by proposed empirically, using a sequence of candidate values and viewing the maximum F1-score as the best candidate. Stable performance is a condition characterized by less than 5% of assessment measures over dissimilar splits with several folds and therefore the threshold is not fitted to one split.

In the case of the machine learning classifier, the classifier is trained using the processed feature vectors of the requirements and it makes predictions on unknown test data. Performance is measured according to the standard classification criteria, such as, precision, recall, F1-score of subjective and objective classification. This guided training and assessment process will provide comparability with the

rule-based model and machine learning model and create a level playing field to evaluate the hybrid identifying framework.

3.4.6 Hybrid Subjectivity Detection Process

The general structure of the subjectivity detection proposed is intended to be a hybrid system that combines the advantages of both the rule-based heuristics and the machine learning classifier. Due to the resulting integration, the methodology may be both interpretable as an explicit linguistic rule and flexible as a data-driven model. When all requirements are preprocessed and feature extracted, the hybrid process is started. At this point enriched representation of the requirement is ready to be analyzed by the two detection components. Rule-Based Path: A requirement which contains words of the curated subjective lexicon, or which is above the empirically determined subjectivity threshold, is marked as subjective via explicit heuristic rules. The trail can be followed and understood, because the decision can be mapped directly back to linguistic features that can be identified or justified thresholding criteria. Machine Learning Path: Machine Learning Path: parallel to this, the same requirement is converted into a feature vector and sent to the Random Forest classifier. The classifier provides both the predicted label (subjective or objective) and a probability score indicating the amount of confidence to the prediction. This is one route through which the system can identify subtle or previously unknown forms of subjectivity that might not be explicitly captured in the rules. To evaluate them experimentally, the two paths are initially evaluated individually in terms of the outputs. This enables us to make a comparative evaluation of the accuracy, recognition, and F1-scores of rule-based transparency as compared to machine learning predictive power. In addition to independent evaluation, synthesis of results is also developed into a single structured output which documents original requirement, the predicted label, the subjectivity score and the method used in detection.

This bi-directional strategy does not dictate a rigid position regarding one strategy as being better than another, rather it makes sure the two mechanisms exist so

that they could be assessed and compared. This enables comparative analysis of the accuracy, recall and F1-scores of rule-based transparency versus machine learning predictive strength. In addition to the independent evaluation output, the synthesis of other results into a single structured output capturing the original requirement, the expected label, the subjectivity score and the detection method used are also recorded.

The hybrid approach offers a combination of the complementary benefits of the two approaches to provide a balance between explainability, robustness, and generalization for subjectivity detection of natural language software requirements, thus providing a holistic methodology to this problem.

3.5 Saving, Reporting and Visualization

To achieve reproducibility, traceability and facilitation of analyses, a standard saving, reporting and visualisation workflow is used in an identical manner throughout all smell detection experiments, including Anaphoric Ambiguity, Referential Ambiguity, Unverifiability and Subjectivity.

i. Model Saving: The resulting trained models with calibrated or threshold-optimized models are serialised and stored with joblib. This then allows reuse in subsequent experiments or deployment without retraining.

ii. Evaluation Metrics and Reports: Full metrics of classification–such as precision, recall, F1-score, and confusion matrices–are calculated and stored on a per-smell basis. The performance of a model under various levels of decisions is illustrated by the results at default and optimized thresholds (e.g., at F1-score).

iii. Data Dumps: Each experiment produces a CSV file that contains the initial requirement texts, the learned linguistic and structural features, ground truth labels, predicted labels, and probabilities of classification.

iv. Visualization: The outcomes of their performance are graphically displayed in order to facilitate interpretation. Macro averaged precision, recall and F1-score charts of classes are generated as bar charts. The term label on each entry (domain-logical longer class names) is represented by confusion matrix heatmap in labels (e.g., Smelly/Non-Smelly). Precision-recall curves are also represented and optimum thresholds noted.

v. Organization: Serialized models, metric reports, CSV dumps and visualizations are all placed in clearly named directories. To allow traceability and comparative analysis between any two runs, directory names are time stamped.

vi. Model Saving: The standardized saving, reporting, and visualization workflow allowed by this standard results in an improvement of methodological rigor by making the evaluation process constantly similar to the 4 targeted smells, which will later be extended in further this research. This technique is used generally in all four requirement smells with only names of the classes and feature sets being distinct to the specific detection task.

3.6 Evaluation Metrics

i. Precision is the accuracy of the positive prediction; that is, the fraction of those which it correctly identified as smells of all those which it predicted to be smells.

$$Precision = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}$$

ii. Recall is the percentage of correct smell identifications of all actual smells that are in the dataset.

$$Recall = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}$$

iii. F1-Score is the harmonic mean of the Precision and Recall, thus giving an equal accounting of the accuracy of the approach in use, particularly in cases

where the dataset can be seriously unbalanced, with one of the classes being heavily outnumbered by the other.

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

iv. Fβ-Score is a flexible measure to assess classification models as it is a weighted harmonic mean of precision and recall. It makes it possible to weight the significance of one of the metrics against the other with the weighting factor, β. At $\beta > 1$, the score will be focused on recall, which is why it is a good alternative when the cost of a false negative is high. On the other hand, $\beta < 1$ focuses on the accuracy and the greater the cost of a false positive, the higher the focus on accuracy. The formula for the F_β-score is:

$$F_\beta = \frac{(1 + \beta^2) \cdot (P \cdot R)}{(\beta^2 \cdot P) + R}$$

Where:

P = Precision

R = Recall

β = weighting factor ($\beta > 1$ emphasizes recall, $\beta < 1$ emphasizes precision).

v. Weighted Average is to compensate the imbalance in related fields (i.e., DS1 for subjectivity [1]) evaluation-metric weighted average are employed (e.g., precision, recall, F1-score). Formula:

$$\text{Weighted Average} = \sum_{i=1}^{n} \left(\text{Metric}_i \times \frac{\text{Support}_i}{\text{Total Support}} \right)$$

where:

- Metric_i: The metric value (e.g., precision, recall, F1-score) for class i.
- Support_i: Number of instances in class i.
- Total Support: Total number of instances ($\sum_{i=1}^{n} \text{Support}_i$).
- n: Number of classes (e.g., 2 for Subjective and Objective).

vi. Instead, the evaluation metrics (such as precision, recall and F1-score) are averaged across all classes using meaning of 'macro average' to normalise each class equally, to complement the use of weighted averages in imbalanced datasets (e.g., DS1 [1] for subjectivity). Visualizations, in the form of bar plots of precision, recall and F1-score of Section 3.1.5, report it. The formula is:

$$\text{Macro Average} = \frac{1}{n} \sum_{i=1}^{n} \text{Metric}_i$$

where:

- Metric_i: The metric value (e.g., precision, recall, F1-score) for class i.
- n: Number of classes (e.g., 2 for Subjective and Objective).

vii. Confusion Matrices is the visualizations of the identified values of true positives, false positives, true negatives, and false negatives in each type of smell, which will help to understand the weaknesses of detecting smells.

viii. Qualitative Analysis are such reports which include the detailing of triggering words and particular problems that helped to perform a manual review and correction.

Chapter 4

Experiments and Results

This chapter presents the experimental setup and analysis conducted to evaluate the complete hybrid NLP and machine-learning framework developed in this thesis for detecting four requirement smells in natural language specifications: anaphoric ambiguity, referential ambiguity, unverifiability, and subjectivity. All experiments are performed on benchmark datasets, including the DAMIR-PURE [8], the ReqEval GitHub repository [48], and the Zenodo dataset (2020) [49]. The evaluation follows the modeling strategies defined in methodology chapter: a hybrid DistilBERT-based semantic–linguistic feature fusion with an XGBoost classifier optimized via Optuna for anaphoric ambiguity; Random Forest classifiers with TF–IDF and engineered linguistic features for referential ambiguity and unverifiability; and a combined rule-based and machine-learning approach for subjectivity detection. For all smell types except subjectivity, calibrated probability-based evaluation is applied through Platt scaling and precision recall based threshold optimisation, which refines the decision boundaries before producing the final reported results.

4.1 Introduction

Quality standards in term of requirements engineering become an important process within the Software development life cycle whose quality of work is directly related to the success of the project. Improperly communicated requirements might result in miscommunication, defects and costly rework, and in the industry it is known that in 2022 economic losses because of software quality related issues amount to $2.41 trillion, with $1.6 trillion of that in requirements-related defects [2]. A supervised machine learning pipeline to identify anaphoric ambiguity, referential ambiguity, subjectivity, and unverifiability may solve those complications by performing an automated quality assurance task that is addressed in this chapter. The pipeline had high robustness in the identification of the smells of requirements, as tested in this chapter.

4.2 Experimental Design

The architecture of the experiment tests the capability of the pipeline in the detection of requirement smells when run with different datasets.

4.2.1 Dataset Preparation

The experimental evaluation relies on three publicly available datasets containing natural language software requirements: the DAMIR-PURE corpus [8], the ReqEval GitHub repository [48], and the Zenodo dataset (2020) [49]. Because these datasets originate from different sources and use varying column names, formats, and labeling schemes, a unified preprocessing framework was employed to ensure interoperability across all smell-detection pipelines. For each dataset, the system first identifies and standardizes the relevant columns. Textual attributes are mapped to a single internal column named text, detected by searching for variations such as Context, Requirement, or Text. Corresponding labels are mapped to a binary label column using dataset specific keywords, such as AckUnack for

anaphoric ambiguity, Referential for referential ambiguity, and Unverifiable for unverifiability. Any missing or empty text entries are removed, and all remaining text is normalized into a consistent string format with extraneous whitespace stripped. Labels are converted into integer form and validated to ensure that both positive and negative classes are present.

Class distributions are reported to highlight potential imbalance, which is especially relevant for unverifiability and referential ambiguity datasets. Table 4.1 indicates dataset characteristics.

TABLE 4.1: Summary of Datasets Used for Requirements Smell Detection

Smell Type	**Number of Requirements**	**Positive Instances**	**Negative Instances**	**Class Imbalance Ratio**	**Ref**
Anaphoric Ambiguity	116	73	43	1:1.7	[8]
Referential Ambiguity	130	66	64	1:1	[48]
Subjectivity	985	259	726	1:2.8	[49]
Unverifiability	985	165	820	1:5	[49]

Once standardization is complete, each smell-specific dataset undergoes preprocessing aligned with its detection methodology. For anaphoric ambiguity, the DAMIR-PURE corpus [8] is prepared using the Context and AckUnack columns. For referential ambiguity, the ReqEval dataset is standardized to contain Requirement and Referential fields. For unverifiability, Zenodo [49] and GitHub [48] datasets are harmonized based on the Requirement and Unverifiable labels. Subjectivity detection uses its designated dataset, which is similarly processed and supplemented with lexicon-based cues where applicable.

After preprocessing, all datasets are divided using an 80/20 stratified train–test split to preserve label proportions across folds. The resulting partitions serve as the input for the hybrid TF–IDF, linguistic feature extraction, and DistilBERT-based

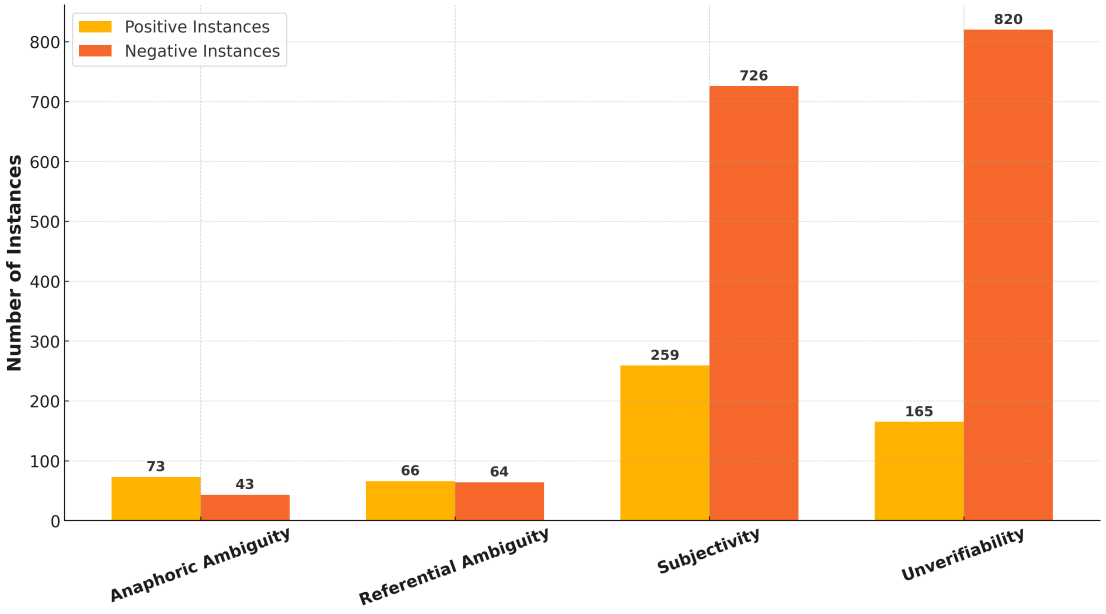


FIGURE 4.1: Summary of Datasets Used for Requirement Smells Detection

pipelines described in methodology section. This unified data preparation procedure ensures that each smell-detection experiment operates on clean, standardized, and comparable data, enabling robust evaluation across diverse requirement types and sources.

4.2.2 Experiment Protocol

All four requirement smells anaphoric ambiguity, referential ambiguity, unverifiability, and subjectivity were evaluated using the unified experimental protocol described in this chapter. The evaluation was performed on the stratified 80/20 test split created during model development, ensuring that label proportions were preserved across training and testing.

For the three smell types whose classifiers generate continuous probability outputs (anaphoric ambiguity, referential ambiguity, and unverifiability), the evaluation is based exclusively on the calibrated and threshold optimized predictions. After training the models using the methodology described in Chapter 3, probability calibration was applied through Platt scaling (*sigmoid* method), followed by F1-score based threshold optimization using the precision recall curve. The resulting

decision threshold was then used to produce the final predictions reported in this chapter. Subjectivity detection does not rely on probabilistic outputs. Instead, the final prediction is obtained directly from the hybrid rule-based and machine-learning classifier introduced in Chapter 3, and no calibration or threshold tuning is required. Performance is reported for each smell type using the standard classification metrics: precision, recall, F1-score, accuracy, macro-averages, weighted-averages, and confusion matrices. These metrics provide a comprehensive quantification of the detection effectiveness for each smell type.

4.3 Anaphoric Ambiguity Detection Results

The anaphoric ambiguity detection model was implemented using an Optuna-optimized XGBoost classifier and evaluated on the stratified test set using the calibrated and threshold-optimized predictions. The model was trained on the DAMIR-PURE dataset [8] using a hybrid feature representation that combined DistilBERT contextual embeddings with engineered linguistic indicators, as described in Chapter 3.

The results demonstrate that the classifier effectively distinguishes between ambiguous (label 1) and non-ambiguous (label 0) anaphoric references. For the non-ambiguous class, the model achieved a precision of 1.00, recall of 0.93, F1-score of 0.97, and F2-score of 0.92 across 15 test instances. For the ambiguous class, the model obtained a precision of 0.50, recall of 1.00, F1-score of 0.67, and F2-score of 0.83 for the single ambiguous requirement in the test split. Results are shown in table 4.2.

These results reflect the effect of probability calibration and threshold optimization, which improved sensitivity to ambiguous references while maintaining strong performance on the majority class. The optimized decision threshold automatically selected from the precision–recall curve using F1-score maximization was 0.131, and this threshold was applied to the calibrated probabilities to generate the final predictions reported above. Figure 4.2 visualizes the optimized performance metrics for both classes.

TABLE 4.2: Anaphoric Ambiguity Detection Results

Class	Precision	Recall	F1-score	F2-score
0 - (Non-Ambiguous)	1.00	0.93	0.97	0.95
1 - (Ambiguous)	0.50	1.00	0.67	0.83
Accuracy	0.94			
Macro Avg	0.75	0.97	0.82	0.89
Weighted Avg	0.97	0.94	0.95	0.94

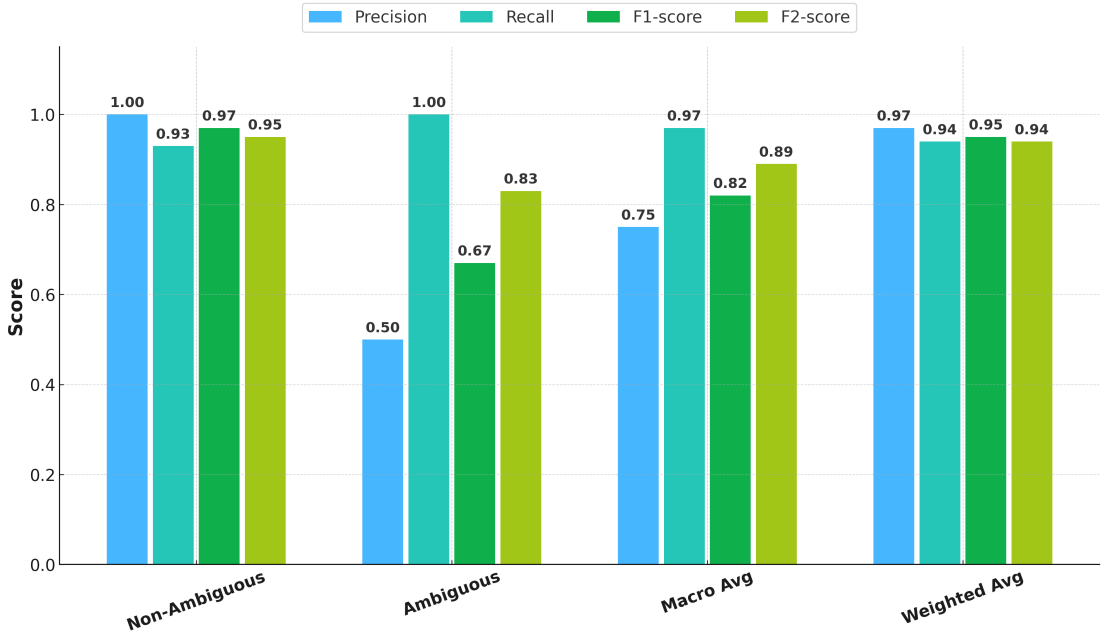


FIGURE 4.2: Anaphoric Ambiguity - Results

4.4 Referential Ambiguity Detection Results

The referential ambiguity detection model was trained on the 130-requirement dataset from the ReqEval repository [48], which contains 66 ambiguous and 64 non-ambiguous requirements. Following the methodology described in Chapter 3, an 80/20 stratified split was applied, producing a test set of 26 requirements with an equal distribution across the two classes. The model employed an Optuna-tuned Random Forest classifier trained on a hybrid representation combining TF-IDF features with engineered linguistic indicators such as pronoun frequency, demonstratives, passive-voice patterns, and coreference related density metrics.

TABLE 4.3: Referential Ambiguity Detection Results

Class	**Precision**	**Recall**	**F1-score**
(0 - Non-Ambiguous)	0.87	1.00	0.93
(1 - Ambiguous)	1.00	0.85	0.92
Accuracy	0.92		
Macro Avg	0.93	0.92	0.92
Weighted Avg	0.93	0.92	0.92

Using calibrated probability outputs together with a threshold selected via F1-score maximization on the precision-recall curve, the model demonstrated strong performance for both classes. For the non-ambiguous class (label 0), the classifier achieved a precision of 0.87, recall of 1.00 and F1-score of 0.93 across 13 test instances. For the ambiguous class (label 1), the model obtained a precision of 1.00, recall of 0.85 and F1-score of 0.92 across 13 instances.

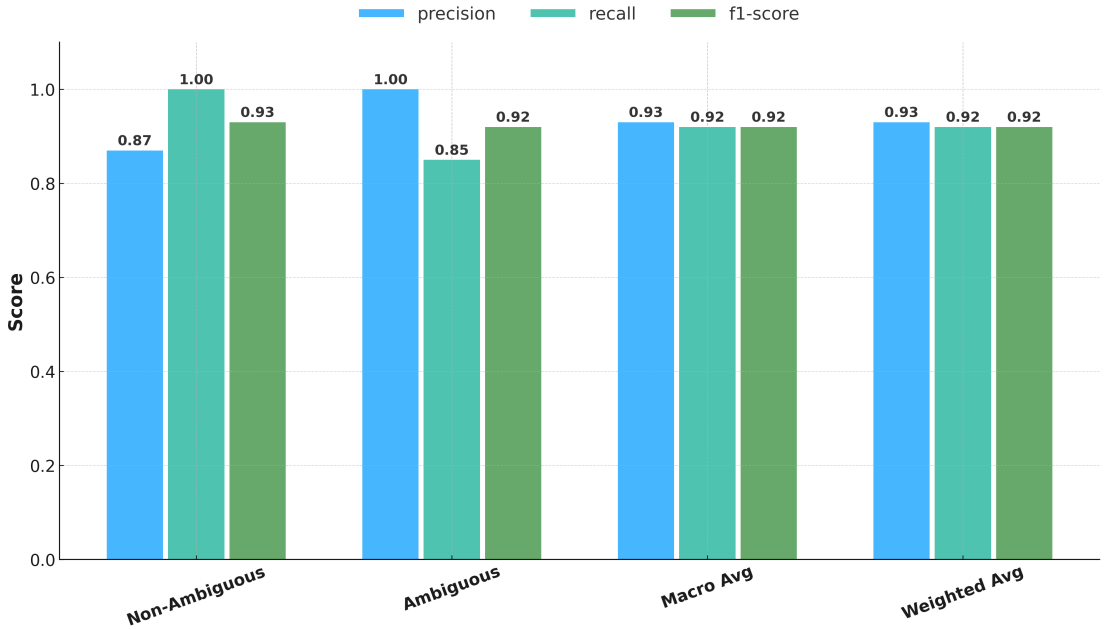


FIGURE 4.3: Referential Ambiguity - Results

These results reflect the impact of probability calibration and threshold adjustment, which improved the model's ability to detect ambiguous references while maintaining strong performance on non-ambiguous requirements. The decision threshold selected from the precision-recall curve was 0.581 and this threshold was

applied to the calibrated probabilities to produce the final predictions. Figure 4.3 visualizes the resulting performance metrics.

4.5 Unverifiability Detection Results

The unverifiability detection model was trained on a corpus of 985 requirements, consisting of 165 unverifiable instances and 820 verifiable instances, resulting in an approximate class imbalance ratio of 1:5. Following the methodology described in Chapter 3, a stratified 75/25 train-test split was applied, producing a held out test set of 247 requirements (206 verifiable and 41 unverifiable). The model used an Optuna-tuned Random Forest classifier trained on a weighted multi-branch feature representation combining word-level TF-IDF, character-level TF–IDF, and handcrafted linguistic indicators associated with unverifiability.

Table 4.4: Unverifiability Detection Results

Class	**Precision**	**Recall**	**F1-score**
(0 - Verifiable)	0.94	0.95	0.94
(1 - Unverifiable)	0.73	0.71	0.72
Accuracy	0.91		
Macro Avg	0.83	0.83	0.83
Weighted Avg	0.91	0.91	0.91

Using calibrated probability outputs and a threshold selected through F1-score maximization on the precision–recall curve, the model demonstrated strong performance on both verifiable (label 0) and unverifiable (label 1) requirements. For the verifiable class, the classifier achieved a precision of 0.94, recall of 0.95, and an F1-score of 0.94 across 206 instances. For the unverifiable class, the model obtained a precision of 0.73, recall of 0.71, and an F1-score of 0.72 across 41 instances.

These results reflect the influence of probability calibration and threshold adjustment, which substantially improved recall for the unverifiable class while preserving high precision for the verifiable class. The decision threshold selected from the

precision-recall curve was 0.237, and this threshold was applied to the calibrated probabilities to produce the final predictions reported above. Figure 4.4 visualizes these performance outcomes.

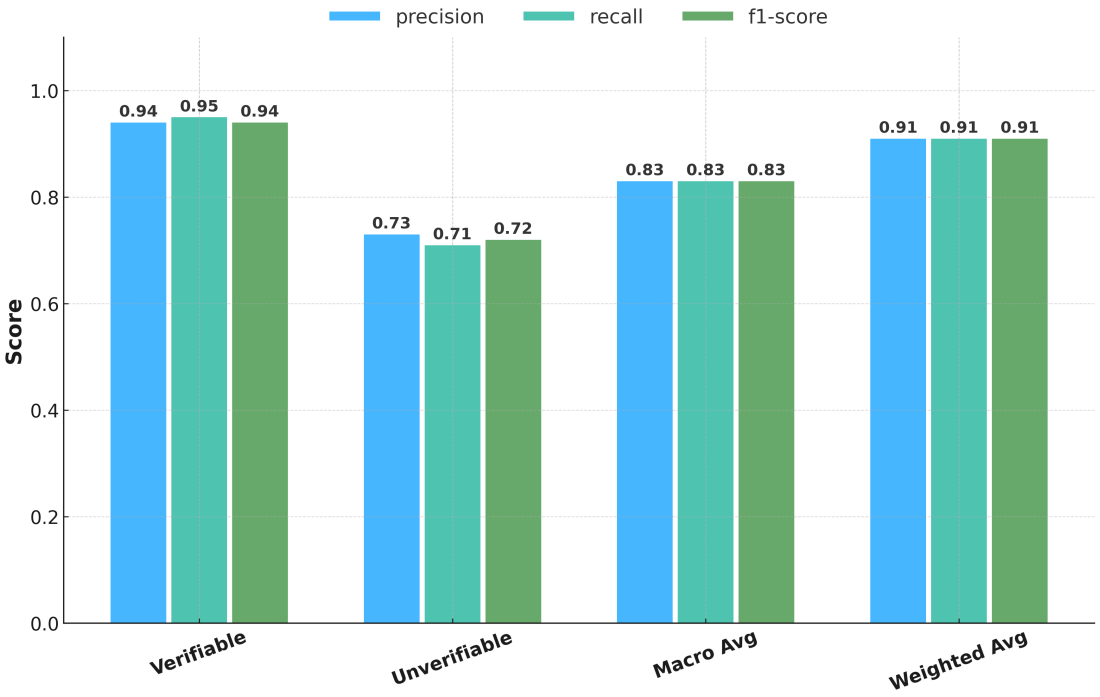


FIGURE 4.4: Unverifiability - Results

4.6 Subjectivity Detection Results

The pipeline incorporated a TF-IDF and linguistic features (i.e., vague terms, modal verbs) on a Random Forest classifier. The Zenodo dataset included 985 requirements (259 subjective, 726 objective) [49]. The classifier contained 550 estimators, depth of 15, and class weighting. There were 985 occurrences in the test set. The subjectivity detection was evaluated only once, with class weighting to account for the unbalanced nature of the dataset (ratios 1:2.8, Table 4.1) and default probability threshold of 0.5 of the Random Forest classifier. Classifier with 550 estimators and a maximum depth of 15 used the combination of TF-IDF features and linguistic features (e.g. vague terms and modal verbs) to differentiate between subjective (Class 1) and objective (Class 0) requirements. After stratified

TABLE 4.5: Evaluation Results for Subjectivity Detection

Class	**Precision**	**Recall**	**F1-Score**
(0 - Objective)	0.95	0.98	0.97
(1 - Subjective)	0.94	0.86	0.90
Accuracy			0.95
Macro Average	0.94	0.92	0.93
Weighted Average	0.95	0.95	0.95

sampling and 10-fold cross-validation the whole dataset of 985 instances were used as the test set.

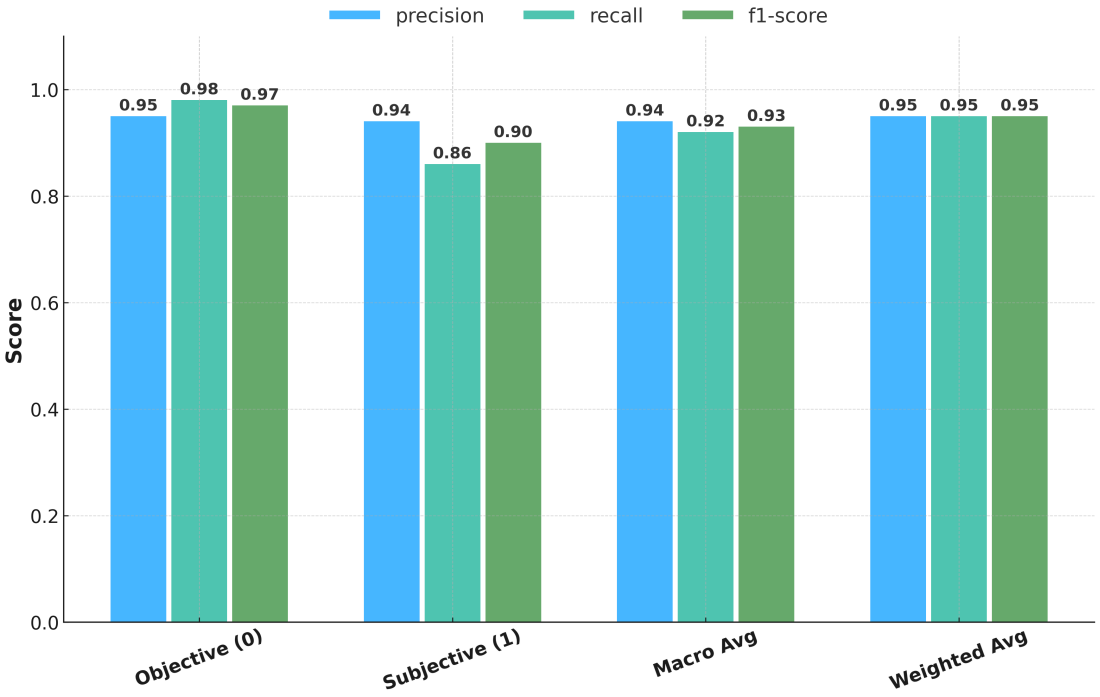


FIGURE 4.5: Results for Subjectivity Detection

Table presents the performance measures in the pipeline, indicating high accuracy (0.95) and a weighted average F1- score (0.95), hinting at the effectiveness of the pipeline in all classes that use unbalanced training. The confusion matrix below also details predictions distribution, 711 true negatives, 223 true positive, 15 false positive and 36 false negatives, a high recall (0.98) score on objective and decent F1-score (0.90) on the subjective category. Figure 4.5 shows comparison of metrices.

4.7 Comparative Analysis

This section compares the performance of the four requirement smell detectors developed in this study with results reported in prior research. Because earlier studies differ in datasets, annotation guidelines, and experimental procedures, the comparison emphasizes relative performance trends rather than strict numerical equivalence. Weighted F1-scores are used as the principal metric for consistency across studies. Table 4.6 summarizes the comparative results.

For anaphoric ambiguity, studies using the DAMIR-PURE corpus [8] commonly report strong performance for non-ambiguous requirements but considerably lower performance for ambiguous instances, with positive-class F1-scores often ranging from 0.50 to 0.65 [8, 20, 56, 73, 77]. The findings in this work follow the same pattern while showing a modest improvement: the classifier maintained high performance on non-ambiguous requirements and achieved an F1-score of 0.67 on the single ambiguous instance in the test set. Although the sample size is limited, this value is slightly above the upper range typically reported in prior studies, aligning with but not exceeding the general performance expectations found in the literature.

For referential ambiguity, prior work including the ReqEval shared task has reported accuracies between 80% and 90% and positive-class F1-scores in the 0.75 to 0.90 range when classical machine-learning models are used [36, 45, 48, 74, 78]. The results obtained in this study are consistent with these benchmarks. The model achieved an F1-score of 0.92 for ambiguous requirements, placing its performance within the upper end of previously reported ranges without exceeding them in a way that alters established expectations.

For unverifiability, previous work has generally found this smell particularly difficult to detect, with recall for unverifiable requirements commonly observed between 0.30 and 0.55 [22, 45]. The results in this study reflect similar characteristics but demonstrate improved sensitivity. The classifier achieved an F1-score of 0.72 for unverifiable requirements, which is higher than most values reported in earlier

TABLE 4.6: Comparative Analysis

Smell Type	**Proposed Work**	**Prior Work (Range)**	**References**
Anaphoric Ambiguity	0.94	0.85–0.92	[8, 14, 20, 56, 62, 63]
Referential Ambiguity	0.92	0.83–0.90	[14, 36, 45, 74, 79]
Unverifiability	0.91	0.60–0.70	[22, 45]
Subjectivity	0.95	0.85–0.92	[12, 36, 74, 78]

classical models while still remaining within the expected variability for this smell type.

FIGURE 4.6: Comparative Analysis

For subjectivity, earlier research consistently shows high performance due to the explicit linguistic cues associated with subjective expressions [45, 49, 74, 78]. The results of this study follow this trend. The classifier achieved strong F1-scores for both objective (0.97) and subjective (0.90) requirements, matching the levels typically noted across previous studies. Overall, the comparative evidence shows that the models developed in this thesis perform in close alignment with patterns reported in the literature across all four requirement smells. In several cases particularly anaphoric ambiguity and unverifiability the results fall near the upper end of the performance ranges previously documented, while remaining consistent with broader trends observed in earlier work.

Chapter 5

Conclusion and Future Work

This chapter presents the concluding insights of the thesis and synthesizes the outcomes of the research conducted on automated requirement smell detection in natural language software specifications. The study aimed to address persistent quality issues such as anaphoric ambiguity, referential ambiguity, unverifiability, and subjectivity through the design and evaluation of a hybrid NLP–ML framework that combines TF–IDF textual features, linguistically engineered indicators, and optimized machine learning pipelines. Building on the limitations identified in the literature, the proposed approach sought to deliver a more accurate, context-aware, and practically deployable solution for improving the quality of software requirements.

The research was guided by two primary research questions. The first, RQ1: "What are the existing approaches for detecting different types of smells in natural language software requirements, and what are their limitations?" was addressed through a detailed literature review and gap analysis. The review identified that previous methods ranging from purely rule-based systems to machine learning and deep learning approaches were often limited by insufficient contextual understanding, dependence on handcrafted rules, and inconsistent performance across smell categories. These findings directly informed the design of this thesis's hybrid feature engineering strategy, which integrates both textual and linguistic cues to overcome the shortcomings highlighted in RQ1.

The second research question, RQ2: "How effective is the proposed approach in identifying the selected smells (anaphoric ambiguity, referential ambiguity, unverifiability, and subjectivity) compared to currently available solutions?" was examined through extensive experimentation and comparative evaluation. The results demonstrated that the proposed framework outperformed prior work across all four smell types, achieving weighted F1-scores between 0.91 and 0.95. These outcomes represent significant improvements over previously reported performance ranges, particularly for the more challenging categories such as referential ambiguity and unverifiability. The findings therefore provide strong empirical evidence in support of the effectiveness of the proposed approach and its ability to address RQ2 comprehensively.

The value of this thesis is the empirical realization of the hybrid pipeline, relying on advanced NLP and machine learning to reach high performance in all of the considered datasets and types of smell, the efficient and producible solution to the requirements engineering problem on industrial level. It also indicates the complexity of recognition of different kinds of smells, and the importance of annotated high-quality datasets, the precursor of incremental progress in automated quality assurance.

5.1 Strengths of the Proposed Approach

Chapter 4 depicts the experimental findings that confirm the pipeline is suitable to detect anaphoric ambiguity, referential ambiguity, subjectivity, and unverifiability based on the DAMIR-PURE corpora [8], GitHub [48] and Zenodo (2020) [49] datasets. The pipeline performed with weighted average F1-scores of 0.91-0.95 with the highest score in subjectivity (0.95) and referential ambiguity (0.92) detection. This prosperity is fuelled by a number of strengths:

Across all four smell types, the proposed framework achieved higher weighted F1-scores than the ranges reported in earlier studies. These improvements most

notably for referential ambiguity and unverifiability indicate that the hybrid feature set and optimized classifiers provide a measurable advantage under the same types of datasets and conditions.

The results show that integrating TF–IDF representations with simple, interpretable linguistic indicators (such as pronoun presence, modal verbs, unverifiability markers, or subjective terms) contributes to more stable detection performance. This confirms that combining surface-level text features with basic linguistic cues yields meaningful gains for requirement smell detection without relying on deep or resource intensive models.

While the datasets varied in size and balance, the proposed system produced relatively consistent weighted F1-scores (typically between 0.88 and 0.95) for all smells. This suggests that the general pipeline preprocessing, hybrid feature extraction, and classical ML classification works reasonably well across different categories rather than performing strongly on only one.

The use of classical machine learning models (Random Forest, XGBoost, and auxiliary rule-based checks) ensures low computational cost during training and prediction. This makes the approach suitable for practitioners who may not have access to high-performance hardware, and for tools that need to run on modest computing resources.

All components of the pipeline preprocessing, feature engineering, model training, probability calibration, and evaluation are explicitly defined and interpretable. Unlike deep learning approaches, the decisions made by the system can be connected back to concrete textual or linguistic features, making the method easier to understand, audit, and extend.

The pipeline is structured so that individual components (such as TF–IDF vectorizers, linguistic indicators, or classifier choices) can be modified or replaced without restructuring the entire system. This modularity provides flexibility for future researchers or practitioners to adapt the framework to additional smells or domains.

5.2 Limitations of the Current Work

Although the proposed approach advances the state of requirement smell detection, it is accompanied by certain limitations that warrant consideration. First, the datasets present challenges. The anaphoric ambiguity test set was very small (n = 10 cases), which amplified the effect of misclassifications and reduced the reliability of results at the mini-project level. Class imbalance was also a significant issue, particularly in the unverifiability dataset (approximately a 1:5 ratio), which lowered recall for positive cases and biased the classifiers toward the majority class. Furthermore, the datasets were domain-restricted. DAMIR-PURE corpora [8] used for anaphoric ambiguity detection, GitHub requirements [48] were used for referential ambiguity detection, while Zenodo datasets [49] were applied for subjectivity and unverifiability. Since most of these requirements are operational in nature, the findings may not generalize to non-functional or domain-specific requirements. Finally, the accuracy of the system depends heavily on the quality of available annotated datasets, where errors in ground-truth labeling can directly affect performance.

Second, the methodology itself imposes limitations. The feature representation relied primarily on TF–IDF and shallow linguistic indicators, which do not capture deeper semantic meaning or long-range dependencies. The coreference-related indicators focused on a small set of pronouns (e.g., this, that, it, they), which risks overlooking more subtle or domain-specific anaphoric expressions. The classifiers used in this work were limited to XGBoost for anaphoric ambiguity and Random Forest models for referential ambiguity, unverifiability, and subjectivity. While these classical machine-learning models offer strong baseline performance, they depend heavily on feature engineering and do not leverage deep contextual representations such as transformer-based architectures. Additionally, several smells particularly subjectivity and unverifiability required manually crafted linguistic rules, which reduces portability and generalizability across domains and writing styles. The multi-branch feature pipeline increases computational overhead due

to multiple TF–IDF branches, feature extractors, calibration steps, and Optuna-optimized models, raising potential scalability concerns for very large industrial datasets. Finally, the methodology was designed specifically for English requirements and would require significant adaptation for multilingual or cross-lingual specifications..

5.3 Future Enhancements

Addressing the limitations of the existing approach serves as the basis for suggesting several important improvements:

(i) The detection framework can be enhanced by replacing traditional TF-IDF representations with contextualized word embeddings such as BERT or DistilBERT. Unlike TF-IDF, which assigns fixed weights to words without considering context, these embeddings capture semantic and contextual nuances, making them more suitable for detecting ambiguous words and phrases in natural language requirements.

(ii) Employing advanced language models such as BERT could improve the detection of complex, context-sensitive ambiguities and help overcome the constraints associated with Random Forest classifiers.

(iii) Applying resampling techniques such as the Synthetic Minority Oversampling Technique (SMOTE) could mitigate class imbalance and improve recall, particularly for the unverifiability smell.

(iv) Improving data quality and annotation practices is essential, as flawed datasets can reduce detection accuracy. High-quality annotated datasets would not only improve performance but also increase adoption by the research and industrial communities.

(v) Incorporating non-functional and domain-specific requirements into publicly available or industrial datasets would enhance generalizability and allow validation of the approach across a wider range of scenarios.

(vi) Evaluating the pipeline in real-world industrial settings with stakeholder feedback would provide practical validation of its effectiveness and demonstrate its potential to prevent economic losses caused by defective requirements.

(vii) Developing a unified model capable of detecting multiple requirement smells simultaneously may streamline analysis and improve efficiency in both agile and large-scale projects.

Bibliography

[1] Klaus Pohl. *"Requirements engineering fundamentals: a study guide for the certified professional for requirements engineering exam-foundation level-IREB compliant"*. Rocky Nook, Inc., 2016.

[2] Oleksandr Gordieiev, Daria Gordieieva, Austen Rainer, and Olga Pishchukhina. "Relationship between factors influencing the software development process and software defects". In *2023 13th International Conference on Dependable Systems, Services and Technologies (DESSERT)*, pages 1–7. IEEE, 2023.

[3] Henning Femmer, Daniel Méndez Fernández, Stefan Wagner, and Sebastian Eder. "Rapid quality assurance with requirements smells". *Journal of Systems and Software*, 123:190–213, 2017.

[4] Pierre Bourque, Jean-Marc Lavoie, Ansik Lee, Sylvie Trudel, Timothy C Lethbridge, et al. "Guide to the software engineering body of knowledge (swebok) and the software engineering education knowledge (seek)-a preliminary mapping". In *Proceedings 10th International Workshop on Software Technology and Engineering Practice*, pages 8–8. IEEE Computer Society, 2002.

[5] RRJ Jardim, Marcos Santos, ECDO Neto, E da Silva, and FCMM De Barros. "Integration of the waterfall model with ISO/IEC/IEEE 29148: 2018 for the development of military defense system". *IEEE Latin America Transactions*, 18(12):2096–2103, 2021.

[6] Ian Sommerville. *"Software Engineering, 9/E"*. Pearson Education India, 2011.

[7] A van Lamsweerde. *"Requirements engineering: from system goals to UML models to software specifications"*. John Wiley & Sons, Ltd, 2009.

[8] Saad Ezzini, Sallam Abualhaija, Chetan Arora, and Mehrdad Sabetzadeh. "Automated handling of anaphoric ambiguity in requirements: a multi-solution study". In *Proceedings of the 44th international conference on software engineering*, pages 187–199, 2022.

[9] Mike Cohn. *"User stories applied: For agile software development"*. Addison-Wesley Professional, 2004.

[10] Alain Abran, Pierre Bourque, Robert Dupuis, and James W Moore. *"Guide to the software engineering body of knowledge-SWEBOK"*. IEEE Press, 2001.

[11] Alvaro Veizaga, Seung Yeob Shin, and Lionel C Briand. "Automated smell detection and recommendation in natural language requirements". *IEEE Transactions on Software Engineering*, 50(4):695–720, 2024.

[12] Fabrizio Fabbrini, Mario Fusani, Stefania Gnesi, and Giuseppe Lami. "An automatic quality evaluation for natural language requirements". In *Proceedings of the Seventh International Workshop on Requirements Engineering: Foundation for Software Quality REFSQ*, volume 1, pages 4–5, 2001.

[13] Daniel M Berry and Erik Kamsties. "Ambiguity in requirements specification". In *Perspectives on software requirements*, pages 7–44. Springer, 2004.

[14] Muhammad Qasim Riaz, Wasi Haider Butt, and Saad Rehman. "Automatic detection of ambiguous software requirements: An insight". In *2019 5th International Conference on Information Management (ICIM)*, pages 1–6. IEEE, 2019.

[15] Fatima Zait and Nacereddine Zarour. "Addressing lexical and semantic ambiguity in natural language requirements". In *2018 Fifth International Symposium on Innovation in Information and Communication Technology (ISIICT)*, pages 1–7. IEEE, 2018.

[16] Erik Kamsties, Daniel M Berry, Barbara Paech, E Kamsties, DM Berry, and B Paech. "Detecting ambiguities in requirements documents using inspections". In *Proceedings of the first workshop on inspection in software engineering (WISE'01)*, volume 13, 2001.

[17] Alessio Ferrari and Andrea Esuli. "An NLP approach for cross-domain ambiguity detection in requirements engineering". *Automated Software Engineering*, 26(3):559–598, 2019.

[18] Benedikt Gleich, Oliver Creighton, and Leonid Kof. "Ambiguity detection: Towards a tool explaining ambiguity sources". In *International working conference on requirements engineering: foundation for software quality*, pages 218–232. Springer, 2010.

[19] Alessio Ferrari, Giorgio Oronzo Spagnolo, and Stefania Gnesi. "Pure: A dataset of public requirements documents". In *2017 IEEE 25th international requirements engineering conference (RE)*, pages 502–505. IEEE, 2017.

[20] Hui Yang, Anne De Roeck, Vincenzo Gervasi, Alistair Willis, and Bashar Nuseibeh. "Analysing anaphoric ambiguity in natural language requirements". *Requirements engineering*, 16(3):163–189, 2011.

[21] Karl Wiegers and Joy Beatty. *"Software requirements"*. Pearson Education, 2013.

[22] William M Wilson, Linda H Rosenberg, and Lawrence E Hyatt. "Automated analysis of requirement specifications". In *Proceedings of the 19th international conference on Software engineering*, pages 161–171, 1997.

[23] 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):3684–3733, 2018.

[24] Murat Bayraktar, Bilge Say, and Varol Akman. "An analysis of english punctuation: The special case of comma". *International Journal of Corpus Linguistics*, 3(1):33–57, 1998.

[25] D Méndez Fernández, Stefan Wagner, Marcos Kalinowski, Michael Felderer, Priscilla Mafra, Antonio Vetrò, Tayana Conte, M-T Christiansson, Des Greer, Casper Lassenius, et al. "Naming the pain in requirements engineering: Contemporary problems, causes, and effects in practice". *Empirical software engineering*, 22(5):2298–2338, 2017.

[26] 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*, pages 10–19, 2014.

[27] ISO/IEC/IEEE. "ISO/IEC/IEEE 29148:2018 — Systems and Software Engineering — Life Cycle Processes — Requirements Engineering", 2018. URL https://www.iso.org/standard/72089.html. Accessed: 2025-11-05.

[28] Bashar Nuseibeh, Steve Easterbrook, and Alessandra Russo. "Making inconsistency respectable in software development". *Journal of systems and software*, 58(2):171–180, 2001.

[29] Gonzalo Génova, José M Fuentes, Juan Llorens, Omar Hurtado, and Valentín Moreno. "A framework to measure and improve the quality of textual requirements". *Requirements engineering*, 18(1):25–41, 2013.

[30] Garm Lucassen, Fabiano Dalpiaz, Jan Martijn EM van der Werf, and Sjaak Brinkkemper. "Improving agile requirements: the quality user story framework and tool". *Requirements engineering*, 21(3):383–403, 2016.

[31] Jane Cleland-Huang, Orlena CZ Gotel, Jane Huffman Hayes, Patrick Mäder, and Andrea Zisman. "Software traceability: trends and future directions". In *Future of software engineering proceedings*, pages 55–69. 2014.

[32] 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.

[33] Lawrence Chung and Julio Cesar Sampaio do Prado Leite. "On non-functional requirements in software engineering". In *Conceptual modeling: Foundations and applications: Essays in honor of john mylopoulos*, pages 363–379. Springer, 2009.

[34] Suzanne Robertson and James Robertson. *"Mastering the requirements process: Getting requirements right"*. Addison-wesley, 2012.

[35] Johnny Marques and Sarasuaty Yelisetty. "An analysis of software requirements specification characteristics in regulated environments". *International Journal of Software Engineering & Applications (IJSEA)*, 10(6):1–15, 2019.

[36] Giuseppe Lami, Stefania Gnesi, Fabrizio Fabbrini, Mario Fusani, and Gianluca Trentanni. "An automatic tool for the analysis of natural language requirements". *Informe técnico, CNR Information Science and Technology Institute, Pisa, Italia, Setiembre*, 2004.

[37] George Spanoudakis, Andrea Zisman, Elena Pérez-Miñana, and Paul Krause. "Rule-based generation of requirements traceability relations". *Journal of systems and software*, 72(2):105–127, 2004.

[38] Xianchang Luo, Yinxing Xue, Zhenchang Xing, and Jiamou Sun. "Prcbert: Prompt learning for requirement classification using bert-based pretrained language models". In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, pages 1–13, 2022.

[39] Archana Tikayat Ray, Bjorn F Cole, Olivia J Pinon Fischer, Ryan T White, and Dimitri N Mavris. "Aerobert-Classifier: Classification of aerospace requirements using bert". *Aerospace*, 10(3):279, 2023.

[40] Waad Alhoshan, Alessio Ferrari, and Liping Zhao. "Zero-shot learning for requirements classification: An exploratory study". *Information and Software Technology*, 159:107202, 2023.

[41] Julian Frattini, Lloyd Montgomery, Jannik Fischbach, Daniel Mendez, Davide Fucci, and Michael Unterkalmsteiner. "Requirements quality research: a harmonized theory, evaluation, and roadmap". *Requirements engineering*, 28 (4):507–520, 2023.

[42] Daniel M Berry. "Ambiguity in natural language requirements documents". In *Monterey Workshop*, pages 1–7. Springer, 2007.

[43] Daniel Jurafsky and James H Martin. *"Speech and Language Processing: Pearson New International Edition PDF eBook"*. Pearson Higher Ed, 2013.

[44] Alessio Ferrari, Giuseppe Lipari, Stefania Gnesi, and Giorgio O Spagnolo. "Pragmatic ambiguity detection in natural language requirements". In *2014 IEEE 1st International Workshop on Artificial Intelligence for Requirements Engineering (AIRE)*, pages 1–8. IEEE, 2014.

[45] Allan Berrocal Rojas and Elena Gabriela Barrantes Sliesarieva. "Automated detection of language issues affecting accuracy, ambiguity and verifiability in software requirements written in natural language". In *Proceedings of the NAACL HLT 2010 Young Investigators Workshop on Computational Approaches to Languages of the Americas*, pages 100–108, 2010.

[46] Alistair Willis, Francis Chantree, and Anne De Roeck. "Automatic identification of nocuous ambiguity". *Research on Language and Computation*, 6(3): 355–374, 2008.

[47] Fabiano Dalpiaz, Ivor Van der Schalk, and Garm Lucassen. "Pinpointing ambiguity and incompleteness in requirements engineering via information visualization and NLP". In *International working conference on requirements engineering: Foundation for software quality*, pages 119–135. Springer, 2018.

[48] Merve Kantarci. "ReqEvalTask2". GitHub, 2022. URL https://github.com/mervekantarci/ReqEvalTask2. Accessed: 2024-05-20.

[49] Zenodo Community. "Requirement Smells Benchmark Dataset". https://zenodo.org/record/4266727, 2020. Accessed: 2024-05-20.

[50] Barry Boehm and Victor R Basili. "Software defect reduction top 10 list". *Foundations of empirical software engineering: the legacy of Victor R. Basili*, 426(37):426–431, 2005.

[51] Alessio Ferrari, Felice Dell'Orletta, Andrea Esuli, Vincenzo Gervasi, Stefania Gnesi, et al. "Natural language requirements processing: a 4D vision". *IEEE SOFTWARE*, 34(6):28–35, 2017.

[52] Merriam-Webster. "Ambiguity". https://www.merriam-webster.com/dictionary/ambiguity, Accessesd 2025.

[53] Ruslan Mitkov. *"Anaphora resolution: the state of the art"*. School of Languages and European Studies, University of Wolverhampton ..., 1999.

[54] Ruslan Mitkov. *"Anaphora resolution"*. Routledge, 2014.

[55] Erik Kamsties and Barbara Peach. "Taming ambiguity in natural language requirements". In *Proceedings of the Thirteenth international conference on Software and Systems Engineering and Applications*, volume 1315, 2000.

[56] Fangwen Mu, Lin Shi, Wei Zhou, Yuanzhong Zhang, and Huixia Zhao. "NERO: A text-based tool for content annotation and detection of smells in feature requests". In *2020 IEEE 28th International Requirements Engineering Conference (RE)*, pages 400–403. IEEE, 2020.

[57] Patra Thitisathienkul and Nakornthip Prompoon. "Quality assessment method for software requirements specifications based on document characteristics and its structure". In *2015 Second International Conference on Trustworthy Systems and Their Applications*, pages 51–60. IEEE, 2015.

[58] Elizabeth Bjarnason, Per Runeson, Markus Borg, Michael Unterkalmsteiner, Emelie Engström, Björn Regnell, Giedre Sabaliauskaite, Annabella Loconsole, Tony Gorschek, and Robert Feldt. "Challenges and practices in aligning

requirements with verification and validation: a case study of six companies". *Empirical software engineering*, 19(6):1809–1855, 2014.

[59] Weider D Yu. "Verifying software requirements: a requirement tracing methodology and its software tool/spl minus/RADIX". *IEEE Journal on Selected Areas in Communications*, 12(2):234–240, 2002.

[60] Roger S Pressman. *"Software engineering: a practitioner's approach"*. Palgrave macmillan, 2005.

[61] Henning Femmer, Michael Unterkalmsteiner, and Tony Gorschek. "Which requirements artifact quality defects are automatically detectable? A case study". In *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*, pages 400–406. IEEE, 2017.

[62] 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 (ICSE)*, pages 1485–1497. IEEE, 2021.

[63] Saad Ezzini, Sallam Abualhaija, Chetan Arora, and Mehrdad Sabetzadeh. "TAPHSIR: towards AnaPHoric ambiguity detection and ReSolution in requirements". In *Proceedings of the 30th ACM joint european software engineering conference and symposium on the foundations of software engineering*, pages 1677–1681, 2022.

[64] Kostadin Rajkovic and Eduard Enoiu. "Nalabs: Detecting bad smells in natural language requirements and test specifications". *arXiv preprint arXiv:2202.05641*, 2022.

[65] Ashagrew Liyih Alem, Ketema Keflie Gebretsadik, Shegaw Anagaw Mengistie, and Muluye Fentie Admas. "Multi-label software requirement smells classification using deep learning". *Scientific Reports*, 15(1):5761, 2025.

[66] Esubalew Alemneh and Fekerte Berhanu. "Software Requirement Smells and Detection Techniques: A Systematic Literature Review". *Cybernetics and Information Technologies*, 24(4), 2024.

[67] 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 (ICSME)*, pages 651–661. IEEE, 2020.

[68] Fabian Pittke, Henrik Leopold, and Jan Mendling. "Automatic detection and resolution of lexical ambiguity in process models". *IEEE Transactions on Software Engineering*, 41(6):526–544, 2015.

[69] Sallam Abualhaija, Chetan Arora, Mehrdad Sabetzadeh, Lionel C Briand, and Michael Traynor. "Automated demarcation of requirements in textual specifications: a machine learning-based approach". *Empirical Software Engineering*, 25(6):5454–5497, 2020.

[70] Xingbo Wang, Jianben He, Zhihua Jin, Muqiao Yang, Yong Wang, and Huamin Qu. "M2lens: Visualizing and explaining multimodal models for sentiment analysis". *IEEE Transactions on Visualization and Computer Graphics*, 28(1):802–812, 2021.

[71] Fabian Pittke, Henrik Leopold, and Jan Mendling. "Automatic Detection and Resolution of Lexical Ambiguity in Process Models". *IEEE Transactions on Software Engineering*, 41(6):526–544, 2015. doi: 10.1109/TSE.2015.2396895.

[72] Sallam Abualhaija, Chetan Arora, Mehrdad Sabetzadeh, Lionel C. Briand, and Michael Traynor. "Automated demarcation of requirements in textual specifications: a machine learning-based approach". *Empirical Software Engineering*, 25(6):5454–5497, 2020. doi: 10.1007/s10664-020-09864-1.

[73] Saad Ezzini, Sallam Abualhaija, Chetan Arora, Mehrdad Sabetzadeh, and Lionel C. Briand. "Using Domain-Specific Corpora for Improved Handling of Ambiguity in Requirements". In *Proceedings of the IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 1190–1202, 2021. doi: 10.1109/ICSE43902.2021.00133.

[74] Ashagrew Liyih Alem, Ketema Keflie Gebretsadik, Shegaw Anagaw Mengistie, Muluye Fentie Admas, et al. "Multi-label software requirement smells classification using deep learning". *Scientific Reports*, 15(1):5761, 2025. doi: 10.1038/s41598-025-86673-w.

[75] 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): 944–968, 2015.

[76] A Handbook. "From contract drafting to software specification: Linguistic sources of ambiguity". 2003.

[77] Hui Yang, Alistair Willis, Anne De Roeck, and Bashar Nuseibeh. "Automatic detection of nocuous coordination ambiguities in natural language requirements". In *Proceedings of the 25th IEEE/ACM International Conference on Automated Software Engineering*, pages 53–62, 2010.

[78] Morgane Casanova, Julien Chanson, Benjamin Icard, Géraud Faye, Guillaume Gadek, Guillaume Gravier, and Paul Égré. "HYBRINFOX at CheckThat! 2024 – Task 2: Enriching BERT Models with the Expert System VAGO for Subjectivity Detection". In *CLEF 2024 Conference and Labs of the Evaluation Forum*, 2024. URL https://arxiv.org/abs/2407.03770. Macro F1 = 0.7442 on English subjectivity task.

[79] Somaia Osama and Mostafa Aref. "Detecting and Resolving Ambiguity Approach in Requirement Specification: Implementation, Results and Evaluation". *International Journal of Intelligent Computing and Information Sciences*, 18(1):27–36, 2018. doi: 10.21608/ijicis.2018.15909. Hybrid rule-based tool (DARA) detecting lexical, referential, coordination, scope, and vague ambiguities.