PRIORITIZATION OF SOFTWARE AND SYSTEM REQUIREMENTS THROUGH NATURAL LANGUAGE PROCESSING FOR TESTING SOFTWARE

Vasco Mascarenhas Paula Bastos Leitão

Mestrado em Ciência de Dados

Dissertação orientada por:

Prof. Doutor Ibéria Vitória de Sousa Medeiros

2021
“Time is the most valuable thing a man can spend.”

— Theophrastus.
Acknowledgements

This dissertation results from a long period of work at home and at the Faculty of Sciences of the University of Lisbon, which would not be possible without the support of my family, colleagues and friends.

First of all, I want to thank my supervisor, Prof. Ibéria Medeiros, who called me for this challenge and in which I could learn and grow every day. Her capacity for work and her determination to teach and learn with her students inspired and motivated me in the most challenging periods. I appreciate her willingness to answer any questions, whether in person or remotely, and her ability to develop productive discussions regarding the work I have done.

To Daran Smalley, a huge thank you for all the meetings and patience to answer my questions about the knowledge and domains tackled in this work. Though we only had the opportunity to chat behind a webcam, it was great to share insights about our everyday life in this weird period of our life.

Throughout these two years in the MSc in Data Science, I had the opportunity to share the Faculty spaces with people dedicated and tireless in sharing knowledge. I owe a big thanks to them, especially to Beatriz Lima, Catarina Gamboa, David Almeida, Gustavo Fonseca, João Castanheira, Rúben Branco and Sofia Amorim, for the several moments of fraternization and mutual help during classes, breaks, preparations for evaluation moments and discussions about the thesis.

A big thank you to my closest friends for listening to me and motivating me to be a curious person with the will to grow: Afonso, Beatriz, Diogo, João, Joana, Margarida, Manuel, and finally, a big thank you to Ana, who accompanied me on this journey every day and who, in many moments, gave me the necessary push to overcome the obstacles imposed by this path.

Last but not least, I want to thank my family for all their support in my academic path: to my parents, Sandra and José Carlos, for teaching me fundamental values that have accompanied me throughout these years; to my siblings, Mafalda and Afonso, for the moments of joy that I share with you and that help me clear my head; to my grandparents, Teresa and Avelino, for their love and concern for their grandson; and to my aunt, Teresa, who always encouraged me to want to follow the path I most wanted.

This work was partially supported by the national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference to LASIGE Research Unit (UIDB/00408/2020), and through P2020 with reference to LISBOA-01-0247-FEDER-039238, XIVT project, an ITEA3 European project (I3C4-17039).
Abstract

Safety-critical systems have been a constant and increased presence in industrial production, such as railways and vehicles. These systems are highly configurable and must be intensively tested by system engineers before being deliverable to customers. This process is highly time-consuming and might require associations between the product features and requirements demanded by customers. Requirement prioritization looks to recognize the most relevant requirements of a system, aiming to reduce the costs and time of the testing process. Machine Learning has been shown useful in helping engineers in this task, automating associations between features and requirements. However, its application can be more difficult when requirements are written in natural language and if a ground truth dataset does not exist with them.

In our work, we present ARRINA, a Natural Language Processing-based recommendation system able to extract and associate components from safety-critical systems with their specifications written in natural language and process customer requirements and map them to components. The system integrates a Weight Association Rule Mining framework to extract the components and their associations and generates visualizations that can help engineers understand which components are generally introduced in project requirements. The system also includes a recommendation framework that can associate input requirements to existing subsystems, reducing engineers’ effort in terms of requirement analysis and prioritization.

We performed several experiments to evaluate the different components of ARRINA over four railway’s subsystems and input requirements. As a result, the system achieved 90% of accuracy, which denotes its importance in reducing the time-consuming of engineers in discovering the correct subsystem links and prioritizing requirements for the testing process.

Keywords: Requirement engineering, software testing, natural language processing, association rule mining, recommendation systems
Resumo Alargado

O processo de testagem de software é uma parte essencial do processo de engenharia de requisitos (ER) para compreender melhor o comportamento dos sistemas e para detectar quaisquer falhas que possam ocorrer. De acordo com o relatório World Quality Report (WQR), há muitos objectivos a atingir quando se trata de garantir a qualidade do software e os testes realizados. Um dos objetivos é reduzir os tempos das várias iterações no processo de testagem, com a intenção de eliminar intervenção humana, diminuir os custos, aumentar a qualidade e, consequentemente, entregar os produtos mais rapidamente aos clientes.

Um sistema de software deve ser devidamente testado durante o seu ciclo de vida de desenvolvimento. Geralmente, a capacidade de um sistema a ser testado depende da cobertura de um conjunto de testes, ou seja, quantos cenários são analisados ou características do sistema estão envolvidas no processo, ou mesmo quantos casos de teste podem ser gerados a partir de cenários ou características do sistema, para a sua devida execução. Em teoria, quanto maior for a cobertura, melhor, embora isto não seja exequível na prática devido a restrições de tempo e custos. Esta é uma tarefa difícil e de carácter complexo, principalmente porque um produto pode ter um grande número de variantes\(^1\) e características. No entanto, dadas as exigências dos clientes, o tempo disponível que os produtores têm até à sua entrega aos clientes não é, na maioria dos casos, suficiente para o testar adequadamente.

A Engenharia de Linha de Produtos de Software (ELPS) é uma área que tenta resolver esta tarefa. As Linhas de Produtos de Software (LPS) são conjuntos de sistemas intensivos em software que partilham séries familiares de características da linha de produtos, que permitem derivar um produto individual de características reutilizáveis da linha de produtos. Estas características satisfazem um grupo de requisitos exigidos pelos clientes, formalmente conhecidos como requisitos de cliente (RC) porque cada característica pode ser associada a vários requisitos padrão. A ELPS apoia a reutilização de características em grande escala, o que leva à poupança de custos e tempo no processo de desenvolvimento do produto, ao mesmo tempo que satisfaz as necessidades dos clientes. Para além disso, um produto é constituído por especificações de concepção do produto, ou especificações de design de produto (ED), as quais definem o funcionamento fulcral do produto. Concretamente, numa ED, o engenheiro fornece informações sobre os componentes envolvidos num sistema, a forma como estes interagem entre si, as suas propriedades e o seu impacto nas funcionalidades do sistema.

Cada entrega de produto pode conter múltiplas configurações, e o teste exaustivo de cada configuração é um processo extremamente demorado, especialmente para sistemas críticos de segurança (por

\(^1\)Configurações diferentes de produto com diversos módulos de software partilhados entre produtos e que partilham um conjunto de características.
exemplo, sistemas automóveis, ferroviários, ou de infra-estruturas). Todo o sistema de software é concebido para lidar com todas as configurações possíveis e não pode ser inteiramente testado. Assim, quando uma configuração é escolhida, o sistema de software é adaptado para satisfazer os seus requisitos. No entanto, este sistema de software final precisa de ser testado de forma exaustiva e adequada, o que é um processo que leva um tempo considerável, como já constatado. Para resolver este problema, antes de o testar, é realizado um processo de priorização de requisitos, com o objectivo de obter os requisitos mais relevantes do sistema que devem ser testados, de modo a lhes dar prioridade.

Os RCs precisam de ser analisados por engenheiros de sistema para estes compreenderem as diferenças dentro de um produto padrão. Este é um processo manual que pode demorar muitos meses, em que o engenheiro deve correlacionar todas as características de um produto e os existentes requisitos de projeto em papel. Para satisfazer estas características, o engenheiro pode escolher entre três opções: (1) Actualizar o produto padrão para uma nova revisão; (2) Adaptar o produto padrão para satisfazer as exigências dos clientes; (3) Modificar os requisitos. No entanto, este processo é susceptível de erros, uma vez que os RCs são descritos em linguagem natural (LN), e cada projecto pode ser ligado com 100 a 2000 requisitos. Além disso, a sua exactidão depende também totalmente da experiência do engenheiro.

Adicionalmente, os RCs podem ser escritos em linguagem natural ou numa linguagem restrita e carecem de qualquer utilização de técnicas de escrita formal, significando que os engenheiros podem escrever requisitos de formas variadas. Soluções para alcançar a priorização e reduzir o esforço do lado do engenheiro podem exigir a identificação e extracção de requisitos. Técnicas como o Processamento de Linguagem Natural (PLN) ou a Recuperação de Informação (RI) podem ser aplicadas para apoiar as tarefas linguísticas apresentadas no campo da ELPS.

O principal objectivo desta tese é criar um sistema que possa recomendar e priorizar EDs a requisitos que antes não estavam ligados a um Subsistema de Propulsão e Controlo (SPC) de um sistema ferroviário. Concentramo-nos na abordagem de um cenário de ELPS centrado na identificação de semelhança e variação de requisitos, utilizando técnicas de PLN e RI, dada a existência de corpora constituída por descrições textuais de EDs e RCs. O sistema extrai EDs e RCs de documentos de SPC, escritos em LN e, destes, extrai componentes relevantes, necessários para realizar métodos de semelhança essenciais para recomendações.

EDs e RCs são escritos em LN e seguem domínios diferentes mas relacionados, ou seja, o primeiro segue um domínio de baixo nível, enquanto o segundo se enquadra num domínio de alto nível. Os EDs são introduzidos como uma descrição abstracta do produto padrão, que descreve a sua implementação. Por outro lado, os RCs são definidos como uma descrição concreta de uma característica de um produto a ser implementado para um cliente. Uma especificação pode estar ligada a vários requisitos, enquanto que um requisito pode estar ligados a várias especificações.

O sistema ARRINA (Association and Recommendation for Requirements in Natural Language) que propomos processa tanto requisitos como especificações para identificação e extracção de componentes, características e informações relevantes. A abordagem do modelo do sistema estabelece uma sequência de métodos de PLN e RI que permite derivar produtos de um LPS, oferecendo uma análise para a integração de novos requisitos e que especificações devem ser associadas a estes.

A partir dos componentes relevantes identificados, o modelo é responsável pela representação do
conhecimento dentro dos domínios e recomendação aos requisitos entre domínios. Estas tarefas são independentes uma da outra. Na representação do conhecimento, o sistema é responsável pela extração de regras de associação e a sua representação através de visualizações de Grafos de Conhecimento para cada domínio. A partir de tais visualizações, um engenheiro pode ter uma visão geral dos componentes relevantes e das suas relações dentro de um domínio. Inicialmente, considerou-se que só seria possível fazer recomendações através da extração de regras de associação. Contudo, uma vez que as EDs e os RCs seguem domínios distintos, explorou-se a recomendação através de métodos de semelhança.

Na tarefa de recomendação, o sistema recorre a métodos de semelhança baseados em tokens para compreender as especificações mais relevantes ligadas a um ou mais requisitos inseridos no mesmo. A partir destas recomendações, um engenheiro pode decidir se uma especificação pode ser ligada a um requisito fornecido ao sistema, reduzindo significativamente o tempo esperado para avaliar as ligações. Propomos uma métrica de similaridade diferente das padronizadas, e também estudamos a razão pela qual técnicas de PLN de estado da arte e baseadas em Aprendizagem Profunda não se enquadram no âmbito do nosso problema.

Tivemos acesso a quatro SPCs e usamos o sistema com cada um deles. Fizemos experiências do modelo com vários passos da sequência de tarefas sobre estes subsistemas e um conjunto de requisitos de entrada que nunca foram ligados a qualquer SPC. Em quase todos os subsistemas, mais de 90% dos componentes principais extraídos foram considerados altamente ou possivelmente relevantes para os engenheiros. A partir destes componentes, podemos definir regras e visualizações que se adequam à estrutura dos subsistemas. A avaliação das recomendações foi realizada através de duas metodologias. Na primeira, avaliámos as recomendações com as ligações que já existem nos subsistemas. Nesta avaliação, obtivemos uma precisão próxima dos 90% em determinados subsistemas, utilizando apenas métodos de semelhança baseados em tokens com os componentes identificados. No segundo quadro, a avaliação dos requisitos de entrada dos transformadores principais, nunca ligados aos subsistemas, foi considerada, pelos engenheiros, boa a muito boa em muitos casos. A partir destes resultados, acreditamos que o nosso modelo tem o potencial de reduzir os custos de teste sem exigir grandes quantidades de dados ou conhecimento relacionado.

Palavras Chave: Engenharia de software, testagem de software, processamento de linguagem natural, extração de regras de associação, sistemas de recomendação
# Contents

1 Introduction  
1.1 Context and Motivation .................................................. 1  
1.2 Objectives ................................................................. 2  
1.3 Contributions ............................................................. 4  
1.4 Document Structure ...................................................... 4  

2 Background and Related Work  
2.1 Requirement Engineering .................................................. 7  
2.2 Information Extraction through Natural Language Processing .......... 9  
2.3 Noun Phrase Chunking ..................................................... 11  
2.4 Association Rule Mining ................................................... 13  
2.5 Sentence Embeddings and Similarity .................................... 15  
2.6 Regression ................................................................. 17  
2.7 Recommendation Systems ............................................... 18  
2.8 Set Similarity .............................................................. 21  
2.9 Final Remarks ............................................................. 23  

3 ARRINA System  
3.1 Problem Definition ....................................................... 25  
3.1.1 Main Description ....................................................... 26  
3.1.2 Challenges and Research Questions ................................... 28  
3.2 Raw Dataset and Basic Concepts ....................................... 29  
3.3 The ARRINA Model ....................................................... 30  
3.3.1 Preprocessing ......................................................... 31  
3.3.2 Main Chunking ....................................................... 32  
3.3.3 Rule and Knowledge Representation ................................ 34  
3.3.4 Recommendation ..................................................... 36  

4 Implementation  
4.1 Preprocessing ............................................................. 45  
4.2 Main Chunking ............................................................. 46  
4.3 Rule and Knowledge Representation .................................... 47
4.4 Recommendation ................................................................. 49
4.5 Main Algorithm ................................................................. 50

5 Evaluation ........................................................................ 53
  5.1 Extracting and Processing Requirements .......................... 54
  5.2 Main Chunk Identification and Relevancy ....................... 54
    5.2.1 Optimal Threshold ..................................................... 55
    5.2.2 Main Chunk Relevancy .............................................. 55
  5.3 Knowledge Subsystem Representation ............................. 57
  5.4 Inter- and Intra-domain Recommendations ...................... 60
    5.4.1 Inter-domain Framework ......................................... 60
    5.4.2 Intra-domain Framework ....................................... 66
  5.5 Key Performance Indicator Analysis ................................. 69

6 Conclusion ........................................................................ 75
  6.1 Future Work ................................................................. 75

References ........................................................................ 77

Appendix A Rule Mining and Visualizations ........................... 85
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Traditional RNN structure</td>
<td>10</td>
</tr>
<tr>
<td>2.2</td>
<td>Noun Phrase Chunking over a sentence</td>
<td>11</td>
</tr>
<tr>
<td>2.3</td>
<td>Item-based collaborative filtering</td>
<td>19</td>
</tr>
<tr>
<td>3.1</td>
<td>Overview of the subsystems and input requirements</td>
<td>26</td>
</tr>
<tr>
<td>3.2</td>
<td>Input and output of the preprocessing task for a design specification</td>
<td>27</td>
</tr>
<tr>
<td>3.3</td>
<td>Example of a customer requirement associated with the design specification</td>
<td>30</td>
</tr>
<tr>
<td>3.4</td>
<td>Overview of the ARRINA model</td>
<td>32</td>
</tr>
<tr>
<td>3.5</td>
<td>Simple depiction of main chunks and their derived chunks</td>
<td>33</td>
</tr>
<tr>
<td>3.6</td>
<td>Similarity between a design specification (top) and a customer requirement (bottom)</td>
<td>36</td>
</tr>
<tr>
<td>3.7</td>
<td>Top-K recommendations for an input requirement</td>
<td>37</td>
</tr>
<tr>
<td>3.8</td>
<td>SBERT architecture for similarity between two sentences</td>
<td>38</td>
</tr>
<tr>
<td>3.9</td>
<td>10-fold cross-validation pipeline</td>
<td>43</td>
</tr>
<tr>
<td>4.1</td>
<td>Class diagram of the recommendation system model implementation</td>
<td>46</td>
</tr>
<tr>
<td>5.1</td>
<td>Knowledge graphs for DSs and CRs of subsystem P1</td>
<td>59</td>
</tr>
<tr>
<td>5.2</td>
<td>Frequent patterns and association rules extracted for each PCS</td>
<td>59</td>
</tr>
<tr>
<td>5.3</td>
<td>Accurate recommendations in each non-specification identification, K = 5</td>
<td>61</td>
</tr>
<tr>
<td>5.4</td>
<td>Comparison of accurate recommendations with main and chunk, K = 5</td>
<td>62</td>
</tr>
<tr>
<td>5.5</td>
<td>Comparison of positive and negative links</td>
<td>63</td>
</tr>
<tr>
<td>5.6</td>
<td>Comparison of non-textual links in each PCS</td>
<td>63</td>
</tr>
<tr>
<td>5.7</td>
<td>Accuracy per values of K ∈ [1, 15]</td>
<td>64</td>
</tr>
<tr>
<td>5.8</td>
<td>Accuracy comparison of IoC metrics for each PCS, K ∈ [1, 15]</td>
<td>65</td>
</tr>
<tr>
<td>5.9</td>
<td>Comparison of accuracy for different main chunking thresholds, K = 5</td>
<td>67</td>
</tr>
<tr>
<td>5.10</td>
<td>Intra-domain Framework in evaluation</td>
<td>68</td>
</tr>
<tr>
<td>5.11</td>
<td>Engineer validation of top-5 recommendations</td>
<td>68</td>
</tr>
<tr>
<td>5.12</td>
<td>Comparison of AnalysisTime and ToolTime and relation with number of specifications</td>
<td>72</td>
</tr>
<tr>
<td>5.13</td>
<td>Comparing AnalysisTime and ToolAnalysisTime analysis time for different values configuration of DS and IReq.</td>
<td>72</td>
</tr>
<tr>
<td>5.14</td>
<td>Parameter variation in ToolAnalysisTime</td>
<td>73</td>
</tr>
<tr>
<td>A.1</td>
<td>Knowledge graphs for DSs and CRs of subsystem P2</td>
<td>87</td>
</tr>
</tbody>
</table>
A.2 Knowledge graphs for DSs and CRs of subsystem $P_3$. ............................. 87
A.3 Knowledge graphs for DSs and CRs of subsystem $P_4$. ............................. 88
List of Tables

2.1 A small set of transactions of a market. ................................................. 14
2.2 User-item matrix $R$, containing missing entries. ................................. 20
2.3 User-item matrix $R$, fully populated. .................................................. 21

3.1 The number of DSs and CRs in each subsystem, before preprocessing. .......... 28
3.2 Subsystem descriptions. ................................................................. 31
3.3 The rules defined in the chunk grammar. ............................................ 33
3.4 Hyperparameters tested in Transformer training. .................................. 39
3.5 Transformer results for batch size $= 4$. .............................................. 40
3.6 Transformer results for batch size $= 8$. .............................................. 40
3.7 Transformer results for batch size $= 16$. ............................................ 41
3.8 Samples for the $IoC_{wgt}$ regression evaluation. ................................... 42
3.9 Regression evaluation for a specification-requirement pair. .......................... 42
3.10 Parameters tested for grid-search in $IoC_{wgt}$ regression. .......................... 43
3.11 Regression scoring for $IoC_{wgt}$ validation. ......................................... 44

5.1 Association between evaluation subquestions and research questions. ............ 53
5.2 Tokens, sentences, and average sentence length before and after preprocessing. .... 54
5.3 Comparison of patterns and rules extracted for different main chunking thresholds. .... 56
5.4 Chunks, frequency, and main chunking extraction for various thresholds. .......... 56
5.5 Main chunk proportions extracted with each grammar, for $P1$, in each validation iteration. 57
5.6 Chunk relevancy distribution for subsystems $P2$, $P3$ and $P4$. .................... 57
5.7 Itemsets and rules extracted in $P1$ with ARM and WARM. .......................... 58
5.8 Accuracy per K threshold. ............................................................... 64
5.9 Comparison of $main_{ioc}$ accuracy for different main chunking thresholds, $K = 5$. .... 66
5.10 Evaluation categories for a specification-requirement link. .......................... 67
5.11 Parameters for $ToolTime$ calculation. ................................................. 71

A.1 Itemsets and rules extracted in $P2$, with ARM and WARM. ........................ 85
A.2 Itemsets and rules extracted in $P3$, with ARM and WARM. ........................ 86
A.3 Itemsets and rules extracted in $P4$, with ARM and WARM. ........................ 86
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>AHP</td>
<td>Analytical Hierarchical Process</td>
</tr>
<tr>
<td>ARM</td>
<td>Association Rule Mining</td>
</tr>
<tr>
<td>CFG</td>
<td>Context-Free Grammar</td>
</tr>
<tr>
<td>CR</td>
<td>Customer Requirement</td>
</tr>
<tr>
<td>DS</td>
<td>Design Specification</td>
</tr>
<tr>
<td>IE</td>
<td>Information Extraction</td>
</tr>
<tr>
<td>IR</td>
<td>Information Retrieval</td>
</tr>
<tr>
<td>KPI</td>
<td>Key Performance Indicator</td>
</tr>
<tr>
<td>ML</td>
<td>Machine Learning</td>
</tr>
<tr>
<td>NA</td>
<td>Numerical Assignment</td>
</tr>
<tr>
<td>NER</td>
<td>Named Entity Recognition</td>
</tr>
<tr>
<td>NL</td>
<td>Natural Language</td>
</tr>
<tr>
<td>NLP</td>
<td>Natural Language Processing</td>
</tr>
<tr>
<td>PCS</td>
<td>Propulsion and Control Subsystem</td>
</tr>
<tr>
<td>POS</td>
<td>Part-of-Speech</td>
</tr>
<tr>
<td>RE</td>
<td>Requirement Engineering</td>
</tr>
<tr>
<td>RNN</td>
<td>Recurrent Neural Network</td>
</tr>
<tr>
<td>SPL</td>
<td>Software Product Line</td>
</tr>
<tr>
<td>SPLE</td>
<td>Software Product Line Engineering</td>
</tr>
<tr>
<td>STS</td>
<td>Sentence Textual Similarity</td>
</tr>
<tr>
<td>WARM</td>
<td>Weighted Association Rule Mining</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

The work tackled in this dissertation intersects the areas of Natural Language Processing (NLP), Data Science and Requirement Engineering (RE). The problem introduced in this dissertation — requirement recommendation and prioritization — belongs to the RE domain, while the model built to extract, collect and refine data is based on NLP and Data Science methodologies.

The following sections describe why requirement prioritization is vital in RE and how NLP can help achieve automation in this process. Later, we define our research goals and state its contributions. Lastly, we describe the structure of the document, giving a summary of each chapter.

1.1 Context and Motivation

The software testing process is an essential part of the RE process to better understand the systems’ behaviour and to detect any failures that can occur. According to the World Quality Report (WQR) [2], there are many objectives when it comes to quality assurance and testing. One of them is to reduce the testing cycle time, intending to avoid human intervention in full measure, decrease costs, increase quality and, consequently, deliver products faster.

A software system must be properly tested during its development life cycle. Generally, the capability of a system to be tested depends on the coverage of the test suite, i.e., how many scenarios are analyzed or system features are involved in the process, or even how many test cases can be generated from scenarios or systems features and properly executed. In theory, the higher the coverage, the better, though this is not achievable in practice due to time and cost constraints. This is a difficult and complex task, mainly because a product can have a large number of variants1 and features, given the demands of the interested customers, and the available time providers have until delivering it to customers is, in most cases, not enough to test it properly.

Software Product Line Engineering (SPLE) is an area that tries to tackle this task. Software Product

1Different product configurations with diverse software modules shared among products and that share a set of features.
Chapter 1 Introduction

Lines (SPL) are sets of software-intensive systems that share familiar series of product line features, which allow deriving an individual product from reusable features of the product line [40]. These features satisfy a group of requirements required by customers, formally known as customer requirements (CR) because each feature can be associated with many standard requirements. SPL supports the reuse of features on a large scale, which leads to cost and time savings in the product development process while meeting customers’ needs. Moreover, a product is constituted by product design specifications, or design specifications (DS), which define the pivotal functioning of the product. This means that, in a DS, the engineer provides information about the components involved in a system and how they interact between each other, their properties and their impact in the system functionalities.

Each product delivery can contain multiple configurations, and intensive testing of each configuration is a vastly time-consuming process, especially for safety-critical systems (e.g., automotive, railway, infrastructure systems). The whole software system is designed to handle all possible configurations and cannot be tested entirely. Thus, when a configuration is chosen, the software system is adapted to meet its requirements. However, the final software system needs to be tested intensively and adequately, which is a process that takes a considerable amount of time, as we already mentioned. To tackle this, before testing it, a process of requirement prioritization is made, aiming to obtain the most relevant requirements of the system that must be tested and ranked, i.e., prioritizing them.

CRs must be analyzed by system engineers to understand differences within a standard product. This is a manual process that can take many months, where the engineer must correlate all features of a product and the existing project requirements on paper. To meet those differences, the engineer can choose between three options: (1) update the standard product in a new revision; (2) adapt the standard product to meet demands; (3) modify requirements. However, this process is prone to errors since CRs are described in natural language (NL), and each project can be linked with 100 to 2000 requirements. Moreover, its accuracy is also totally dependent on the engineer’s experience.

Additionally, requirements may be described in natural or a restricted language and lack any use of formal writing techniques, as stated in a recent survey from Kassab et al. [35], meaning hence that engineers can write requirements in different manners. Solutions to achieve prioritization and reduce effort on the engineer’s side could require requirement identification and extraction. Techniques such as NLP or Information Retrieval (IR) can be applied to support the linguistic tasks presented in the SPLE field [74].

1.2 Objectives

The main goal of this thesis is to create a system that can recommend and prioritize DSs to requirements that were not before linked to a Propulsion and Control Subsystem (PCS) of a railway system. We focus on tackling an SPLE scenario centered on identifying requirement similarity and variability, using NLP and IR techniques with the inexistence of a corpus composed of DSs and CRs. From PCS documents written in NL, the system extracts DSs and CRs and, from these, retrieves relevant components, required
to perform similarity methods essential to recommendations.

DSs and CRs are written in NL and follow different, but related domains, i.e., the former follows a low-level domain, while the latter fits on a high-level domain. DSs are introduced as an abstract description of the standard product, which describes its implementation and functionalities. On the other hand, CRs are defined as a concrete description of a feature of a product to be deployed for a client. A specification may be linked to various requirements, as well as a requirement may be linked to various specifications.

The ARRINA (Association and Recommendation for Requirements in Natural Language) system we propose processes both requirements and specifications for identification and extraction of relevant components, features and information. The ARRINA model establishes a pipeline of NLP and IR methods that allows deriving products from a SPL, offering an analysis for the integration of new requirements and which specifications to be associated with those.

From the relevant components identified, the model is responsible for the knowledge representation within domains and for recommendation to requirements inter-domains. These tasks are independent of each other. In knowledge representation, the system is responsible for extracting association rules and representing them through knowledge graph visualizations for each domain. From these visualizations, an engineer can overview the relevant components and their relations inside a domain. At first, we thought we could perform recommendation only through association rule mining. However, since DSs and CRs follow distinct domains, we had to explore recommendation through similarity methods.

In the recommendation task, the system resorts to token-based similarity methods to understand the most relevant specifications linked to one or more input requirements. From these recommendations, an engineer can decide if a specification can be linked to a requirement provided to the system, reducing significantly the time expected to correlate and evaluate pairings. We propose a similarity metric different than the ones established as standard, and also study why deep learning, state-of-the-art NLP techniques did not fit into the scope of our problem and the restrictions associated to it.

We had access to four PCSs and executed the system with each one. We experimented the model with variations in some steps of the pipeline over these subsystems and a set of input main transformer requirements that were never linked to any PCS. In almost all subsystems, more than 90% of the main components extracted were considered highly or possibly relevant to engineers. From these components we can define rules and visualizations that suit the structure of the subsystems. Recommendation evaluation was performed in two frameworks. In the first, we evaluated recommendations with the links that already exist in the subsystems. In this evaluation, we obtained accuracy close to 90% in some subsystems, only using token-based similarity methods with the components identified. In the second framework, evaluation of the recommendation of never linked input main transformer requirements to subsystems was considered by engineers as good to very good in some cases. From these results, we concluded that our model has the potential to reduce testing costs without requiring large amounts of data or related knowledge.
This work is part of XIVT\textsuperscript{2} (\textit{eXcellence in Variant Testing}), a ITEA3 Call 4 and P2020 project, that aims to improve the testing process of industrial and highly configurable products. It integrates the knowledge-based requirements process in Work Package 2 (WP2) and was developed over the Use Case from Bombardier Transportation, a member of the Alstom Group\textsuperscript{3}.

1.3 Contributions

The main contributions of this work, contained in ARRINA, the recommendation system we propose, are the following:

- The analysis of CRs and DSs written in NL in intra- and inter- domains;
- A NLP pipeline that accurately extracts relevant components from DSs and CRs;
- A weighted association rule mining method to identify component associations inside the PCS and to represent the knowledge it contains, through graph visualizations;
- A recommendation system based on token-based similarity methods that can help engineers mapping requirements to specifications, and to reduce the testing time;
- Experimental evaluations providing assessments for the three aforementioned thesis results with different configurations over the various PCSs and the set of main transformer requirements.

This research led to the publication of the paper \textit{SRXCRM: Discovering Association Rules Between System Requirements and Product Specifications} in the 4th Workshop on Natural Language Processing for Requirements Engineering, co-located with the 27th International Working Conference on Requirement Engineering: Foundation for Software Quality [38].

1.4 Document Structure

Besides the chapter we are currently in, this dissertation is divided in five more chapters:

- Chapter 2 - Background and Related Work
  This chapter focuses on addressing the foundational concepts required to understand the NLP and IR techniques used in this work and the research related to the problem tackled in it.

- Chapter 3 - Recommendation System
  In this chapter, we define the problem to be solved along with the challenges and research questions arising from these, the raw datasets (PCSs) understood as the problem input, and the model we
proposed to solve the problem. We also introduce some assessments considered vital to make some of the decisions regarding the model.

• Chapter 4 - Implementation
  We state, throughout this chapter, the implementation details in the various algorithms defined along the pipeline of the model. It is also described a class diagram of the implementation.

• Chapter 5 - Evaluation
  This chapter presents the experiments we made to evaluate the model. Experiments were divided into four main parts: (1) results regarding the extraction and processing of specifications and requirements; (2) evaluation of various parameters in component identification and relevancy; (3) findings on the knowledge representation task; (4) results on requirement recommendation, with two different evaluation frameworks; and (5) assessment in Key Performance Indicators of this problem.

• Chapter 6 - Conclusion
  Lastly, we discuss the assets of this work, its threats to validity and propose future research that can follow up from our contributions.
Chapter 2

Background and Related Work

This chapter presents the concepts required to understand better the techniques and methodologies explored in this dissertation. Also, it gives an overview of relevant related work developed. This chapter is divided into eight subsections: 1) Requirement Engineering; 2) Information Extraction through Natural Language Processing; 3) Noun Phrase Chunking; 4) Association Rule Mining; 5) Sentence Embeddings and Similarity; 6) Regression; 7) Recommendation Systems; and 8) Set Similarity. While the first subsection is required to understand the challenge this dissertation tackles better, the others introduce the theory necessary to comprehend the solution introduced in chapter 3. We also give some final remarks about topics described throughout the various subsections of this chapter and their impact in the solution we created.

2.1 Requirement Engineering

RE is described as a part of software engineering concerned with the precision of requirement specifications, their evolution over time and their relation with other specifications [72]. Specifications are often described as informal and behavioural observations, written in NL; hence translation to mathematical specification languages is needed. A requirement is an abstract conceptualization of the system, a condition needed to achieve a specific goal, satisfying a formal document.

RE follows a set of activities that depend on the type of system being constructed and the stakeholders (e.g., clients, developers, users) involved. Activities may include [50]:

- **Requirement elicitation**: engineers are responsible for the gathering, interpretation and deep analysis of requirements that solve specific tasks.

- **System modelling**: generation of abstract descriptions of requirements that are discussed with stakeholders.
• **Requirement specification and validation**: the creation of Requirement Specifications (RS), artifacts that can vary in formality but are focused on delivering a set of descriptions crucial to the system development.

• **Requirement management**: analysis of the evolution of requirements over time, through traceability techniques.

Engineers should aim for completeness in requirement specification, especially in safety-critical systems, in domains such as automotive, maritime, railway, among others. However, most projects have requirements that cannot be fully processed due to time and cost constraints. In practice, one of the most critical aspects of RE is requirement prioritization [11], which is an integral part of the requirement specification and validation activity. Requirement prioritization tries to identify which requirements are the most valuable, providing support to many activities in the project, deciding the core requirements in a system, assessing the risk of not processing a requirement, or even understanding the trade-off between feasibility and cost.

There are multiple traditional approaches to requirement prioritization, such as Analytical Hierarchy Process (AHP) [11] or Numerical Assignment (NA) [11]. In AHP, all possible requirement pairs are compared to understand which is the highest-ranked through a comprehensive overview of requirements. However, the total number of comparisons to perform is $n \times (n - 1)/2$, for $n$ requirements, which results in a high increase of comparisons as $n$ increases. NA is based on grouping requirements into priority groups. Stakeholders are responsible for associating requirements to these groups.

In recent years, NLP models have been quite relevant in RE, especially in requirement generating and modelling tasks. For example, Bertolino et al. propose PLUC [12], focused on deriving requirements in Product Lines. Their work proposes additions to use cases with tags to represent relations between requirements in controlled NL. This allows test case generation with PLUTO [12], based on Category-Partition [51]: use cases act as high-level functions where tags are the possible choices for the various categories in each UC, reducing manual effort in the testing process. However, PLUTO has no requirement prioritization algorithm associated and our work process requirements with a lower formality level.

Some works rely on Machine Learning (ML) and NLP in terms of requirement extraction and prioritization. Mu et al. propose EFRF [48], a framework focused on extracting functional requirements from software requirement specifications by analyzing their linguistic structure. Although this is an interesting approach to NL, the authors do not specify how the model deals with domain-specific terms. Sonbol et al. [64] have come up with ReqVec that focuses on a semantic representation of functional requirements in NL, based on an Information Extraction (IE) pipeline and the Word2Vec embedding model. Semantic representation allows efficient detecting of related requirements given their vector representations, providing a clear structure for categorization. Perini et al. [53] defined a prioritization method named CBRank that combines stakeholders preferences with boosting methods and pairwise elicitation in order to reduce the manual input without losing estimation accuracy. Schlutter et al. [62] propose a pipeline that extracts concepts and relations from requirement documents, which are represented in knowledge
graphs for helping engineers in knowledge detection and querying. However, it is affirmed that this pipeline lacks quality criteria to support validation. We look for validation with constant communication with domain experts to understand if the extraction of terms ensures completeness.

Abbas et al. [4] defined a process focused on requirement reuse analysis, using existing CR in order to recommend features to implement new, unseen CR. Their work is based on techniques such as TFIDF or Doc2Vec, followed by clustering of requirement vectors created by these to then aid the recommendation of new ones. The process had good results in accuracy and requirement identification. However, the lack of understanding of the algorithm from expert users is mentioned recurrently, reducing the acceptance of the methodology. Our work aims at creating a system based on chunking and Association Rule Mining, which is easy to understand and provides precise results about relationships between components. We use an IE pipeline similar to the work of Sonbol et al. [64], but we look for different approaches in prioritization.

Therefore, we focus on delivering a ML approach to NL requirements and specifications in this work to perform requirement extraction and prioritization.

2.2 Information Extraction through Natural Language Processing

IE [32] is defined as a process that focuses on turning unstructured information into structured data through a preprocessing pipeline with NL (obtainable from text) as input. Structured data is essential to extract the meaningful features that will be further processed through ML techniques.

Tokenization is the first step on the preprocessing pipeline, where the text to be processed (a set of sentences written in NL, e.g., English) is split into tokens (the words in a sentence). Next, the Stopwords Removal task removes words that occur very commonly in a NL (e.g., is, it or our in English), since they do not constitute relevant meaning [60] and could represent noise to the quality of the tokens extracted. Finally, each token goes through a Part-Of-Speech (POS) Tagging task, where it is categorized/tagged as a word class, such as a noun, a verb, adverb, among others, for a specific language, based on a corpus (a large set of texts, annotated with tags from a well-defined tagset) that the tagger learns previously. Tagging is an important task, given that lemmatization and chunking algorithms depend directly on the labels learnt, and each word can have multiple possible part-of-speech tags, depending on the context of the sentences where they occur. For example, the word book can be tagged as being a noun or a verb, depending on the inherited context within the sentence.

There are different techniques for POS Tagging, such as Hidden Markov Models (HMM) [32], Average Perceptron Models [67] [20], and deep learning models, as the Recurrent Neural Networks (RNN) [32] [14] [70], Long Short Term Memory (LSTM) or Gated Recurrent Units [19].

Another relevant task in the IE pipeline is the process of Named Entity Recognition (NER) [32], where Named Entities such as persons, organizations, or temporal expressions are identified. Lample et al. [37] proposed a model with a bidirectional LSTM that can capture long-term dependencies with excellent
results in terms of standard evaluation. Although this is a relevant task that could be analyzed, there is no corpora that contains the terminology of the components introduced in this problem.

There has been extensive research focused on reducing the limitations of traditional models in the IE field. The use of deep learning models has been proven to generate state-of-the-art results in various NLP tasks.

Mikolov et al. [47] proposed novel architectures for word representations in vector space. The Continuous Skip-Gram and Continuous Bag-of-Words (CBOW) models follow a similar structure of RNN (Figure 2.1), but resort to a projection layer instead of using a non-linear hidden layer, reducing complexity to log-linear order without losing accuracy. While the CBOW model uses context to predict a single word, the Skip-Gram model predicts context given a single word. Thus, these models are capable of improving IE tasks.

NER is highly dependable on lexicons and external knowledge that refines entity recognition (e.g., locals and institutions). Generally, word embeddings do not use this type of knowledge to generate predictions. Passos et al. [52] seek to use a variation of the Skip-Gram model that can extract information from lexicons in order to improve representation.

Huang et al. [30] examined different variations of LSTMs, in order to have access to more features without losing robustness. These models, mainly the latter, produce excellent results without requiring any word embeddings. However, these models require a very high amount of features, training sets with a high number of samples, and their fitting also has a high computational cost.

The type of data examined in specific domains can be limited in terms of information to extract. Recently, models have been built to explore NER in limited data as a retrieval task [27] through user interaction. In these models, expert users identify entities in a set of Conditional Random Fields (CRF)-ranked sentences, generating training data suitable for NER. This is a feasible approach for specific domains with narrow markings and shows matching between user annotation and domain knowledge. Some adaptations to open IE to specific domains have been made lately as well, in order to reduce user annotation and manual training. For example, Soderland et al. [63] have classified domain arguments from elements extracted in extensive noun phrases, given a small training set. They also proposed an algorithm for rule
extraction that attempts to evaluate the generalization of relations between distinct classes, although most of the relations extracted are verb-centred.

2.3 Noun Phrase Chunking

Noun Phrase Chunking [13] is a type of shallow parsing where chunks that correspond to individual noun phrases are extracted, selecting various subsets of POS tagged tokens available. For extraction of chunks, it is required context-free grammar (CFG) [5] with a set of rules that define the noun phrases to be discovered. These rules will segment a sentence into chunks.

For example, a chunk can consist in “a possessive pronoun followed by a noun”, and therefore each subset of tagged tokens within a sentence that follows this sequence is recognized as a chunk. This recognition is illustrated in Figure 2.2 for the sentence in its first row. Firstly, it is tagged by a POS Tagger (as shown in the second row), following the Penn Treebank tagset [1], a corpus with a large number of English sentences with their words annotated with the type of their syntactic structure (e.g., noun, verb), being each type represented by a tag from a tagset previously defined. For this sentence, the tagset indicates that a possessive pronoun is tagged as PRP$, a singular noun is tagged as NN, and a verb is tagged as VBZ. Next, the noun phrase chunking parser will use the grammar rules to identify the subset of tokens which tags match the sequence <PRP$, NN> (the chunk we want to retrieve). ”My dog” and ”his food” are the two token sequences that match the tag sequences, and so they are recognized as noun phrases (NP), as shown in the last row of the figure.

As described by Abney [5], this process is divided in two phases - chunker and attacher - which we describe below.

1. **Chunker**, considered a non-deterministic left-right parser. Usually, a left-right grammar is a CFG that does not generate any next-action conflicts, and that can be parsed deterministically. Actually, parsers will have to deal with conflicts since a word can be related to many categories. A chunker simulates the non-determinism through configurations or computation snapshots. Each configuration can lead to various actions, and the chunker is responsible for creating a task for each action.

A task is a tuple in the form of <current configuration, next action, score>, where
next action indicates which rule on the CFG will be chosen. score indicates the likelihood of a task leading to the best results, is determined by multiple factors, where lexical frequency is the most impactful. The current configuration contains a control stack, a parse stack, and the current position in the sentence.

For example, analyzing a sentence that starts with the word Water by a grammar defined as:

```
Chunk  −→  NP
Chunk  −→  VP
NP     −→  N
VP     −→  V
```

where the last two lines, respectively, denote that each recognized noun is a noun phrase (NP) and each recognized verb is a verb phrase (VP), and the first two lines indicate that either NP or VP are accepted as chunks.

The chunker is responsible for shifting (SH) words into the parse stack until it recognizes a rule and reducing (RE) the sequence in the stack into a single node. Follow the water example, it creates a queue with two tasks (Formula 2.1).

\[
(([], [], 0), [SH \text{ water}\_N[NP \rightarrow N\bullet]], s_1) \\
(([], [], 0), [SH \text{ water}_V[VP \rightarrow V\bullet]], s_2)
\]  

(2.1)

In both tasks, the first element of the tuple indicates the starting configuration: an empty control stack, an empty parse stack, and the current position in the sentence (0). The second element indicates which action to be taken: either water is recognized as N (noun) or V (verb), i.e., both contain the rule associated with the word class, where the • marks how much of the rule is already seen. The last element indicates the task’s score (\(s_1 > s_2\)). The task with the best score is the first in the queue.

The chunker chooses the best-scored task in the queue and executes the action defined in its tuple. In this case, it will recognize water as a noun, producing a new configuration and a new task, which will contain the result of the applied task. Thus, the queue is now represented by the tasks depicted in Formula 2.2.

\[
((NP \rightarrow N\bullet], [N], 1), [RE NP \rightarrow N], s_3) \\
(([], [], 0), [SH \text{ water}_V[VP \rightarrow V\bullet]], s_2)
\]  

(2.2)

The new task has the following structure and information:

- Current configuration:
(a) Control stack with the action that was triggered
(b) Parse stack with the word class of *water*
(c) Current position in the sentence
   - Next action, to reduce a noun as a noun phrase, since the chunker has reached the end of the rule
   - Score of the task

Since $s_3 > s_2$, the next action is to reduce *water* as NP.

2. *Attacher*, responsible for keeping the validation of the syntax, by linking chunks with syntactic arcs. It works similarly to the chunker, resolving non-deterministic conflict between chunks, a complex task that is not achievable using only context-free parsing. The attacher requires the lexical properties of each word, which are constructed within a frameset. For example, a specific word might have a [DP< ?, PP*, CP>], meaning that it could take an optional direct object (?), any number of prepositional phrases (PP*), and a final embedded clause (CP). This type of morphology is not accessible within chunker implementations and is not the focus of our work, so it will not be detailed in this paper.

The advantage of using chunkers is that only two elements are required as input: a textual description (that follows the preprocessing stated in Section 2.2) and a grammar with rules to define chunks, described through regular expressions. Many state-of-art frameworks, such as NLTK [13] or SpaCy [3], offer implementations that follow these requirements.

### 2.4 Association Rule Mining

The main objective of Association Rule Mining (ARM) [8] is to find relevant relationships between items in large datasets in order to aid decision-makers in a variety of tasks.

ARM introduces the concept of *transaction*. A transaction $t$ is defined as a binary vector that contains the occurrences of a set of items. If an item $k$ occurs in $t$, $t[k] = 1$, otherwise $t[k] = 0$. An association rule takes the form of $\{I_m, ..., I_k\} \implies \{I_j\}$, being $\{I_m, ..., I_k\}$ an itemset and $I_j \notin \{I_m, ..., I_k\}$.

Representing an association rule with the general $A \implies B$ form, where $A = \{I_m, ..., I_k\}$ and $B = I_j$, and being $A \implies B$ defined by an antecedent $A$ and a consequent $B$, such as $A \cap B = \emptyset$, in the ARM field we tend to look for rules that follow the following two metrics:

- **Minimum transactional support $S$:** $A \cup B$ is present in at least $s\%$ of the transactions in the dataset.
- **Minimum transactional confidence $C$:** at least $c\%$ of the transactions that satisfy $A$ also satisfy $B$ (Formula 2.3).

$$C(A \implies B) = \frac{S(A, B)}{S(A)} \quad (2.3)$$
Table 2.1: A small set of transactions of a market.

<table>
<thead>
<tr>
<th></th>
<th>Milk</th>
<th>Onion</th>
<th>Nutmeg</th>
<th>Beans</th>
<th>Eggs</th>
<th>Yogurt</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Rules that satisfy minimum \( S \) and \( C \) are considered strong rules, i.e., the rules that meet \( A \cap B = \emptyset \) condition.

There are other statistical relevant metrics, which derive from the two above. They are:

- **Lift \( L \)**: indicates how much more often the antecedent and consequent of a rule occur together than we would expect if they were statistically independent [16]. If \( L = 1 \), there is no association between \( A \) and \( B \). If \( L < 1 \), there is a tendency for aversion of \( A \), given \( B \). If \( L > 1 \), there is a tendency for the inclination of \( A \), given \( B \). Formula 2.4 shows the definition of \( L \).

\[
L(A \implies B) = \frac{S(A, B)}{S(A) \times S(B)} \tag{2.4}
\]

- **Conviction \( C_v \)**: indicates how much there is a dependence between the antecedent and the consequent. A high conviction means that consequent is highly depending on the antecedent. If items are independent, then \( C_v = 1 \). Formula 2.5 shows how \( C_v \) is calculated.

\[
C_v(A \implies B) = \frac{1 - S(B)}{1 - C(A \implies B)} \tag{2.5}
\]

Values far from 1 in \( L \) and \( C_v \) may show interesting rules.

A well-known example in the ARM field is the *market basket analysis* technique, where each transaction contains a set of products (items). For example, Table 2.1 describes a small set of five transactions, one per line, composed of a maximum of six items (columns 2-7).

In this example, we will look at the rule \( Eggs \implies Beans \) and the main metrics: *support* and *confidence*.

- \( S(Eggs, Beans) = 0.8 \), since the items occur together in four out of five transactions;
- \( C(Eggs \implies Beans) = \frac{S(Eggs, Beans)}{S(Eggs)} = \frac{0.8}{0.8} = 1 \). Given \( C \), a customer that buys eggs is guaranteed to buy beans.

This process can be done for each possible combination of items in this set of transactions.

The process of creating rules for any possible set of items in large databases can be quite expensive, so before the rule extraction process, first, we need to search for *frequent* itemsets, i.e., itemsets that occur...
in at least $s\%$ of the transactions. Many algorithms, such as Apriori [7], ECLAT [71] or FP – Growth [29], extract the large itemsets in order to reduce computational costs in terms of the ARM process.

Text Mining and ARM are deeply related, bringing an opportunity to represent knowledge in forms that allow quick and simple evaluation. Kulkarni et al. [36] have created a system that supports knowledge discovery on collections of web documents. Rule generation is based on the Generating Association Rule Using Weighting Scheme (GARW), in which keywords have associated weights based on their term frequency, calculated from TF-IDF. First, the top keywords in terms of weight and term frequencies are stored, and those with support higher than a minimum threshold are extracted. Further processing follows the standard ARM framework. Systems such as FACT [25] and EART [43] also take relevant approaches for association rule extraction over keywords in web documents.

ARM has also been associated with classification tasks in the text mining area. Some proposals explore frequent pattern mining on extracted keywords as a comparison benchmark over Naive Bayes [54] and Decision Tree [34] algorithms. These approaches do not use a fully structured NLP pipeline like ours since there is no active search for relevant entities but frequent words instead, reducing the evaluation quality.

In terms of rule creation, different approaches from original ARM have been explored, such as the work of Tao et al. [65], where elements have different weights in the item space through the creation of weighting attributes. This approach is relevant because the downward closure property still holds through adaptations in support and confidence. Our work tries to adapt this framework [65] to the context of the problem described in Section 1. This approach (Weighted Association Rule Mining) is explored in Section 4.

2.5 Sentence Embeddings and Similarity

One of the most relevant challenges in NLP is sentence representations, which can be used in various tasks. Sentence Textual Similarity (STS) is one of these tasks, where the goal is to understand the semantic similarity over a set of documents.

STS can follow a variety of methodologies [44]:

* Knowledge-Based Methods: possible semantic or ontological relationship between words is considered. The knowledge-based similarity is categorized into the following approaches:

1. **Node-based Approach**: each concept is associated with an IS-A taxonomy that carries other concepts. The similarity between two concepts depends on the information shared in common, generally by edge counting methods. This reasoning allows for semantic similarity through information content [58].

2. **Edge-based Approach**: the number of edges between two concepts in a taxonomy is compared directly. The shorter the path between two concepts, the more similar these are.
This reasoning allows for semantic similarity through information content [58].

- **Corpus-Based Methods**: similarity is learned and estimated from a corpus. Relevant approaches include Latent Semantic Analysis, where contextual information is extracted from a large corpus. First, sentences are represented into vectors through a Document-Term Matrix, which makes the correspondence between a set of sentences \( c_1, c_2, ..., c_n \) of a document and each existent term to be found in each sentence. Afterwards, Singular Value Decomposition is performed to reduce dimensionality. Finally, similarity can be measured through the cosine between two sentence vectors.

- **String-Based Methods**: a variety of metrics can be applied to analyze the character or string similarity, such as Damerau-Levenshtein, Jaro, Sorensen-Dice Index, Jaccard Index, among others.

Recently, the methodologies above have evolved to perform similarity over sentence embeddings, obtained through state-of-the-art algorithms that do not require recurrence or convolutions. Vaswani et al. [66] propose the Transformer architecture, which is based on attention mechanisms. Generally, neural models require an encoder-decoder structure. The Transformer model follows this type of architecture, using stacked multiheaded self-attention in the encoder and the decoder. While RNN suffers when understanding terms far inside the sequence, the self-attention mechanism can understand long-range dependencies since all positions in the input sequence are fully connected.

One of the most referenced models that evolve from the Transformer architecture is BERT [23], which pre-trains deep bidirectional representations from the Transformer architecture. BERT is pre-trained in a large corpus of unlabeled text extracted from sources such as Wikipedia (2.5B words) and BooksCorpus (800M words). Instead of only reading a sequence from left to right, BERT understands all the sequences simultaneously through two strategies: Masked Language Model and Next Sentence Prediction. RoBERTa [41] removes the Next Sentence Prediction approach and introduces dynamic masking, which changes in each training epoch and therefore improves the pre-training process. DistilBERT [61] is considered an approximated version of BERT, with almost less 40% of the parameters, running 60% faster and keeping the performance. Sentence-BERT or SBERT [57], is a modification of the BERT network with siamese networks and triplet networks, reducing computation time and making it more viable for the STS task.

Other architectures include Universal Sentence Encoder (USE) [17] that is also built over a Transformer network and trained on the Stanford Natural Language Inference (SNLI) dataset. Most recently, Wang et al. proposed TSDAE [69], where the encoder learns sentences associated with noise and produces a fixed-size vector, which will force the decoder to produce relevant embeddings.

Generally, sentence embedding techniques are evaluated with data from SemEval [21], the STS benchmark, and the SICK-R dataset, through Pearson or Spearman correlation. Pre-trained models such as USE and SBERT generally outperform other methods in these benchmarks since these can use labelled data from specific domains. However, one critic to be made is that STS benchmark performance might not correlate well to task performance in reality [69].
In this work, we annotate data, perform the STS task with SBERT and see how much the results correlate with engineers’ annotations.

2.6 Regression

Regression [26] is a well-known ML task that consists in learning a function estimator \( \hat{f} : X \rightarrow \mathbb{R} \) from a set of examples \( (x_i, f(x_i)) \). Regression differs from classification problems because the former is based on an infinite set of targets, while the latter targets a low-resolution variable divided into various classes. Generally, the goal is not to match the precision of \( \hat{f} \) entirely since some target values in the set of examples might occur due to fluctuations that models cannot capture.

Models are evaluated through the application of a loss function, which objective is to approximate to zero. This function is calculated with residuals \( f(x) - \hat{f}(x) \), there is the difference between the actual value of the function for \( x \) in vector space and the value for the estimator function for the same point.

There are many types of regression established in the ML world. Linear regression is the most simple model, and the model looks to calculate the coefficients \( w \) and \( b \), for \( \hat{y} = w \ast x + b \), minimizing the mean squared error (the loss function).

Ridge regression [31] is a linear model where coefficients are calculated assuming a high correlation between independent variables stated in the regression problem. The model aims to reduce the coefficients \( w \) to zero, giving them low magnitude to target prediction and restricting the model not to overfit - this is called regularization.

Bayesian linear regression [31] estimates a probabilistic model for regression problems through prior distributions. One of the models is the Bayesian Ridge, in which the prior for \( w \) is calculated with spherical Gaussian (Equation 2.6), where Gamma distributions define the prior for \( \lambda \).

\[
p(w|\lambda) = N(w|0, \lambda^{-1}I_p)
\]  
(2.6)

Other relevant regressors can include tree-based regressors that implement Decision Trees in a similar fashion to the classification problem to predict continuous values; and the Support Vector Regressor, which also looks to use regularization to reduce coefficients while at the same time giving some tolerance to error and to data points that do not fall in the trend of the real function. These regression methodologies are addressed further in this document.

In this work, we perform regression not as an end-goal problem of the recommendation system but as a technique to assert the accuracy of the similarity metrics we introduce in the model.
2.7 Recommendation Systems

Recommendation systems are information filtering systems that help users to find items of interest extracted from sizable catalogues. Such systems can be applied in multiple contexts [55], from book recommendation, e-commerce, software or industrial domains [46]. These systems are essential to support and improve decision making, filtering relevant features from users to obtain accurate suggestions. Recommendation techniques employed by these systems can follow different approaches:

- **Collaborative Filtering:** This method is based on recorded interactions between users and items. From these interactions, it is possible to recommend similar items or users. The higher the number of users and interactions, the more accurate the recommendations become. However, this methodology can suffer from a “cold start problem” since it is not feasible to make recommendations if a knowledge base contains only a few interactions.

Collaborative Filtering is divided into two categories:

1. **Memory-based techniques**, where users are characterized into a vector space, being a vector built from the items recorded in interactions of a user. Memory-based CF is achieved through user-based and item-based (Figure 2.3) approaches. User-based techniques recommend items that are popular among the k-nearest-neighbours of a user, while item-based techniques perform similarity between items: two items are strongly similar if users interact identically with these.

2. **Model-based techniques** that resort to Machine Learning or Data Mining models and user-item interactions to perform and improve new recommendations. Learning algorithms include Association Rule Mining, clustering methods such as K-Means or Self-Organized Maps, and Artificial Neural Networks. Since user-item interaction matrices tend to be sparse, these techniques generally require matrix factorization algorithms to decompose these matrices into smaller matrices corresponding to user and item representations.

- **Content-based Filtering:** This method is considered the most adequate when handling specific domains, since it is based only on the attributes of the items. Recommendations are defined by the content of the items already evaluated by a user; items that are highly correlated to those positively rated are recommended. Models such as Decision Trees, TF-IDF, or Neural Networks can design relationships between inputs.

Content-based Filtering is not affected by the same problems as Collaborative Filtering since it does not require other information about user preferences to output recommendations. Moreover, it has a great capacity to adjust to new preferences, adapting recommendations to the new information obtained. However, to provide accurate recommendations, it is necessary to have detailed descriptions about the domain to build relevant features.
One of the relevant uses of the item-based Collaborative Filtering technique is the top-N recommendation problem [22]. These systems are prevalent due to their simplicity and because performance can be quickly evaluated with metrics, such as accuracy, precision and recall [28]. The algorithm requires a \((n \times m)\) user-item matrix \(R\) and a parameter \(k\) that specifies the amount of item-to-item similarities stored in each item to build a matrix \(M\) used to define recommendations. Similarity functions include:

- **Cosine-Based Similarity**, where each item is mapped into a vector space of users, and therefore, the similarity between two items \(i\) and \(j\) is defined as the cosine of \(n\)-vectors that correspond to two columns of \(R\), as described in Equation 2.7.

\[
sim(i, j) = \frac{\vec{R}_{*j} \cdot \vec{R}_{*i}}{\|\vec{R}_{*i}\|_2 \cdot \|\vec{R}_{*j}\|_2}
\]  

(2.7)

- **Conditional Probability-Based Similarity**, where similarity is based on the conditional probability of associating an item \(j\) given that an item \(i\) was already associated, \(P(j|i)\). This measure is less used since it is frequency-based. Also, \(P(j|i) \neq P(i|j)\) does not account for the users with different amounts of items associated.

Applying the recommendation algorithm requires the matrix \(M\) described above, a one-dimensional binary vector \(U\) that contains the items associated with a user \(u\), and the number \(N\) of recommended items. The output is a vector \(v\) with the top-\(N\) items recommended. \(v\) is calculated in two phases:

1. \(v\) is computed as \(M \times U\): entries with a value different than zero correspond to the union of the \(k\) most similar items for each item already associated to \(u\);

2. Entries corresponding to items already associated with \(u\) are set to zero, and entries with values smaller than the \(N\) largest values in the vector are also set to zero.
Table 2.2 denotes a simple example of a user-item matrix $R$ with ratings scaled from 1 to 5, where the item-based collaborative filtering can be applied.

<table>
<thead>
<tr>
<th></th>
<th>$i_1$</th>
<th>$i_2$</th>
<th>$i_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u_1$</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>$u_2$</td>
<td>5</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>$u_3$</td>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>$u_4$</td>
<td></td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

First, similarity in all item pairs is computed. Then, for each pair $(i_u, i_v)$, the model searches the users that rated both elements to map each item to the vector space and perform similarity.

Cosine-based similarity on $(i_1, i_2)$, $(i_2, i_3)$, $(i_1, i_3)$ is computed as described in Equation 2.8.

$$sim(i_1, i_2) = \frac{(5 \times 2) + (3 \times 3)}{\sqrt{5^2 + 3^2} \sqrt{2^2 + 3^2}} = 0.9$$

$$sim(i_2, i_3) = \frac{(3 \times 1) + (2 \times 2)}{\sqrt{3^2 + 2^2} \sqrt{1^2 + 2^2}} = 0.869 \quad (2.8)$$

$$sim(i_1, i_3) = \frac{(2 \times 3) + (3 \times 1)}{\sqrt{2^2 + 3^2} \sqrt{3^2 + 1^2}} = 0.789$$

After extracting pair similarities, it is possible to retrieve the rating for an non-rated item $i_u$ and a user $u_i$. For example, the rating of $i_2$ for $u_1$ is calculated in Equation 2.9:

$$r(u_1, i_2) = \frac{r(u_1, i_1)sim(i_1, i_2) + r(u_1, i_3)sim(i_3, i_2)}{sim(i_1, i_2) + sim(i_3, i_2)}$$

$$= \frac{(2 \times 0.9) + (3 \times 0.869)}{0.9 + 0.869} = 2.49 \quad (2.9)$$

After calculating the ratings for other unknown user-item pairs, and with the matrix $R$ fully populated (Table 2.3), it is possible to perform recommendations based on the new ratings.

Although computational complexity to build the matrix $M$ is defined as $O(m^2n)$, the matrix $M$ tends to be sparse since targeted knowledge bases contain a small number of interactions. To perform a recommendation to a user with $t$ items associated, complexity is given by $O(kt)$.

Such methods might require some tuning to measure similarity accurately, and parameters do not generalize well between different datasets and domains. Models such as SLIM [49] or FISM [33] try to implement learning models that learn objective functions to build the item-to-item matrix. Other models, such as IGMC, look to learn graph patterns based on ratings with graph neural networks [73], providing
Table 2.3: User-item matrix $R$, fully populated.

<table>
<thead>
<tr>
<th></th>
<th>$i_1$</th>
<th>$i_2$</th>
<th>$i_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u_1$</td>
<td>2</td>
<td>2.49</td>
<td>3</td>
</tr>
<tr>
<td>$u_2$</td>
<td>5</td>
<td>2</td>
<td>3.43</td>
</tr>
<tr>
<td>$u_3$</td>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>$u_4$</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

state-of-the-art results in some tasks. Datasets widely used to evaluate performance include MovieLens, data from the Netflix challenge, Douban, among others.

Our work looks to perform recommendations with a comparison framework between software design specifications and requirements, based on textual features, performing the feasible evaluation with metrics such as accuracy and recall.

2.8 Set Similarity

Set similarity techniques look to compute all pairs of similar sets between two collections. Two sets are considered similar if their overlap exceeds a certain threshold. Formally, for two collections $A$ and $B$, a set similarity function $\text{sim}(a, b)$ between two sets $a$ and $b$ and a threshold $t$, is defined as it follows in Equation 2.10.

$$A \triangleright B = \{(a, b) \in A \times B \mid \text{sim}(a, b) \geq t\}$$ (2.10)

Set similarity falls into the token-based algorithms that compare cardinality and features such as intersection and union. Thus, there are multiple metrics for set comparison, and next, we present the ones we considered relevant for our work.

- **Jaccard Index** (Equation 2.11) compares members of two sets to see which are shared and which are distinct. Its value is defined as the division between the size of the intersection and the size of the union of the sets. Index range between 0 and 1, and the higher the index, the higher the similarity between sets. Although it is considered an easy-to-understand index, it does not account for the size of sets and might become overestimated.

$$\text{sim}_\text{jaccard}(a, b) = \frac{|a \cap b|}{|a \cup b|}$$ (2.11)

- **Sørensen–Dice Index** (Equation 2.12) accounts for cardinality between the two sets, equaling twice the number of common elements, divided by the sum of the cardinality of each set. Index range between 0 and 1, and the higher the index, the higher the similarity between sets.
\[
sim_{\text{dice}}(a, b) = \frac{2 \times |a \cap b|}{|a| + |b|}
\]  

(2.12)

- **Tversky Index** (Equation 2.13), a generalization of the Sørensen–Dice Index and the Jaccard Index and two parameters \( \alpha, \beta \geq 0 \). If \( \alpha = \beta = 1 \), the Jaccard Index is obtained. If \( \alpha = \beta = 0.5 \), the Sørensen–Dice is produced.

\[
sim_{\text{tversky}}(a, b) = \frac{|a \cap b|}{|a \cap b| + \alpha|a - b| + \beta|b - a|}
\]  

(2.13)

- **Cosine Index** (Equation 2.14), which can be considered an adaptation of Equation 2.7 to token-based calculations.

\[
sim_{\text{cosine}}(a, b) = \frac{|a \cap b|}{\sqrt{|a| \cdot |b|}}
\]  

(2.14)

Token-based similarity measures are generally normalized to take into account set sizes [45]. There has been some work on improving these metrics, performing evaluation empirically and in some datasets: Likavec et al. propose the Sigmoid similarity measure [39], based on the Jaccard Index; Wang et al. [68] propose fuzzy-token similarity functions that combine token-based and character-based techniques. Azevedo et al. propose \( \text{IoC}_{\text{wgt}} \) [9] (Equation 2.15), a contribution metric between two sets, calculated as the weight of the shared attributes of indicators of compromise in two sets. This metric proposes a measure of the degree of similarity even if the two sets have distinct sizes, something not achievable with the Jaccard Index.

\[
\text{IoC}_{\text{wgt}}(a, b) = \frac{|a \cap b| \times (|a| + |b|)}{2 \times |a| \times |b|}
\]  

(2.15)

Since data collections can have many set pairs to analyze, plenty of research has been developed on computation efficiency for set similarity joins. One of the most used techniques is **prefix filtering** [18], a pruning method that for each pair \((a, b)\), analyzes a ordered subset (a \( \pi \)-prefix) of each element. If the \( \pi \)-prefixes between \( a \) and \( b \) have no intersection, the pair can be pruned since token-based measures rely on the set intersection. The prefix size depends on the threshold and the measure used.

Another technique used to pair pruning is **length filtering**. For a specific set, the size of the possible partners must be in a specific size interval that depends on a threshold \( t \).

Many efficient algorithms [45] use these techniques to crop sets and reduce time and computation costs. However, since the input of our work is domain-restricted subsystems and not massive amounts of pairs, these algorithms are not the focus of our work and will not be detailed in this dissertation.
2.9 Final Remarks

The model described in the following sections carries concepts reported in this section. We intend to present a system able to process requirements with a lower formality level than the ones described in [12], described in NL with little to no writing restrictions. Furthermore, we look for term validation with constant communication with domain experts and engineers, ensuring completeness in the process. To have methodology acceptance, our work aims to present a system based on techniques such as IE, similar to the pipeline defined in [64], and Noun Phrase Chunking (to give an overview of the existing components in the subsystems), ARM (to provide results about relationships between components) and Similarity measures, presenting the output through top-K recommendations, in similar fashion to item-based recommendations described by Deshpande et al. [22]. We also annotate data, performing the Semantic Similarity task with state-of-the-art methodologies, providing a comparison benchmark between these and the Similarity measures described in Section 2.8.
Chapter 3

ARRINA System

Chapter 3 presents ARRINA (Association and Recommendation for Requirements in Natural Language), the model defined in our work to perform requirement recommendation and prioritization through NLP techniques, IE, Noun Phrase Chunking, and ARM. The chapter is split into the following sections:

- Problem Definition (Section 3.1)
- Raw Datasets and Basic Concepts (Section 3.2)
- The ARRINA Model (Section 3.3)

In Section 3.1 we describe the problem of processing NL features in a railway PCS and why the automation of this process can be so valuable to reduce the number of software features and testing time while keeping accuracy. The main objective of this section is to introduce the reader to the problem itself and the variables we tackle, which should be kept in mind when solving this problem.

Section 3.2 presents the four subsystems used as inputs (our raw datasets) and that sustain the problem. We state why these are so distinct, introduce why we need to parse them and define a standard format necessary for the model described in Section 3.3. Also, the section introduces the basic concepts required for a good understanding of the subject tackled in this work.

The last section presents the solution created to resolve the problem, i.e., the ARRINA model. We design a system that resorts to noun phrase chunks to recommend new requirements to subsystems. It also provides a rule-based knowledge graph that gives an overview of the components existing in each subsystem, their connections, and the dependencies between them.

3.1 Problem Definition

The first subsection presents the main description of the problem, while the second states the challenges that arose from it and conduct the work of this thesis.
### 3.1.1 Main Description

A PCS is a safety-critical system consisting of *software design specifications* (PCS standard product feature which we designate for short by DS, or simply specifications) and *customer requirements* (revision of a product feature which we designate for short by CR, or simply requirements) provided by various stakeholders. Specifications and requirements are both written by engineers in NL and belong to two different but related domains. Since new configurations can be derived from these features and require considerable time for testing them properly, it is essential to prioritize requirements to test them later and swiftly. The model to create should be able to link new requirements to the existing subsystems, i.e., to specifications.

Regarding the writing of DSs and CRs, one aspect to be aware of is that writing standards rarely are employed. CRs can be written by different engineers from distinct teams, while DSs and CRs generally follow different domains, though with some intersection of terminology.

A PCS also has a set of Main Transformer requirements. Contrarily to DSs and CRs, these requirements are not linked to any subsystem and require empirical evaluation from domain experts. However, within the scope of the problem, they can be interpreted just the same as CRs. Hence, our objective is that the framework provides subsystem recommendations to the main transformer requirements based on textual features. Again, these requirements were not previously linked to any subsystem.

A high-level overview of a PCS is depicted in Figure 3.1, where we can observe the three elements presented above and their connections with subsystems (e.g., \(P_1\)). The *input requirements* are the new product configurations that we want to link to subsystems. The Main Transformer requirements are isolated into PCS, and, as we also want to link them to subsystem, these are also treated as input requirements.

![Figure 3.1: Overview of the subsystems and input requirements.](image-url)
Figure 3.2a shows an example of a software design specification present in one of the subsystems that are handled in this dissertation. Right away, we can identify these specifications as being composed of declarative sentences that refer to subsystem functioning.

In 7 car configuration shall the valid bits for Motor Converter wheel speed 1 and 2 be set FALSE (there are no motors connected to Motor Converter in this configuration).

(a) Example of a design specification.

(b) POS tokens extracted from the design specification.

Figure 3.2: Input and output of the preprocessing task for a design specification of a subsystem.

Since we are discussing NL descriptions full of domain-specific terminology, the model we aim to define must employ IE and NER techniques that do not refer to generic domains without utilizing text corpora. These descriptions also require cleaning and formatting in order to be correctly processed for knowledge extraction. Generally, these descriptions contain terminology that eases component identification. For example, uppercase terms such as "MC" or "AC", or more complex terms like "high voltage system" or "mechanical braking" can be recognized as components.

Another point of discussion is the subsystem structure. Each DS can be linked to multiple CRs, having high coverage, while many DSs are not linked to any CR, as shown in Figure 3.1.

It is also of engineers’ interest to be provided more relevant information about the existing subsystems, how components are linked between them and the relationships inside subsystem configurations. Our proposal looks to define graph visualizations that can be built from rule mining techniques.

Another important aspect we must consider is that a specification or a requirement might have non-textual elements in its description, such as figures, tables, equations, or even pseudocode snippets. These elements might limit the model’s accuracy since we do not have access to many of these.

Given all issues described above, it is essential to understand which is the best methodology to capture the domain-specific terms and the uniqueness in each subsystem. Therefore, our work intends to experiment with different configurations for noun phrase chunking, rule mining, and semantic similarity to determine which best serves the model.
3.1.2 Challenges and Research Questions

In this subsection, we enumerate the main challenges that should be noted and are tackled by the model. We look to answer them in the following main questions, each one grouping sub-questions:

RQ1. Since the amount of source data to analyze can be very limited and written in NL, can this information be processed without using any related data sources? The amount of data that we work on is small and limited, as described in Table 3.1, column 2. This makes the discovery and generalization of patterns difficult. Usually, there is no access to tagged data (i.e., a corpus or a lexicon) since we are working with text from domain-specific technical sources. We propose an IE pipeline resorting of NLP methods that, instead of recognizing entities based on a NER task that depends on a tagged corpus, extracts relevant information from a semantic and contextual point of view. The information extracted, recurring to noun phrase chunking techniques, is evaluated by domain experts and engineers with knowledge of the subsystems processed.

<table>
<thead>
<tr>
<th>PCS</th>
<th>Object Identifier</th>
<th>DS</th>
<th>CR</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>526</td>
<td>526</td>
<td>164</td>
</tr>
<tr>
<td>P2</td>
<td>192</td>
<td>192</td>
<td>284</td>
</tr>
<tr>
<td>P3</td>
<td>227</td>
<td>227</td>
<td>68</td>
</tr>
<tr>
<td>P4</td>
<td>165</td>
<td>165</td>
<td>120</td>
</tr>
</tbody>
</table>

RQ2. What type of information will be extracted, and can it be used as an input to other testing models? In the IE pipeline, we look for relevant components that later can be used to generate interesting patterns, such as an unsupervised learning process. However, plenty of raw text may not constitute any relevant information, so the pipeline must differentiate it from relevant data and discard it. The data patterns of interest will be used to prioritize requirements assessed in the testing process. This recognition of relevant components with the absence of a corpus allows for the use of CFGs for their identification. CFGs can be easily configured, requiring part-of-speech tagged sentences as input.

RQ3. What is the most suitable way to represent component relations inside each subsystem? Inside subsystems with hundreds of specifications and requirements, engineers might have an interest in understanding how components interact, i.e., if component A has more connections with component B than with component C, or which is the component with more connections inside the subsystem. We intend to define visualizations from existing knowledge bases that provide an overview of relations between components through graph visualizations built from association rule mining.
RQ4. Can we link prioritize requirements using information extracted from DSs and CRs? DSs and CRs follow different domains. Given the demands from customers for new product configurations, new CRs will appear and can be integrated into a system, which requires association with existing DSs. An essential task in our proposal is to find out how to link these objects, specifying the most suitable DSs to be linked to a new CR through recommendation techniques. To reduce the heavy load on the engineers’ side, our model will look into recommendation techniques and analyze whether a recommendation based on textual features can provide accuracy and precision.

RQ5. How is the evaluation performed, and which are the frameworks used? To understand if the model is accurate and precise on the recommendations provided, we look into two evaluation frameworks: (1) inter-domain framework, for each subsystem, we recommend the CRs already linked to the existent DSs. Accuracy and precision are based on the correct associations extracted and given recommendations; (2) intra-domain framework, we recommend Main Transformer requirements to the subsystems we work with, and system engineers are responsible for evaluating if the matching is correct or not.

3.2 Raw Dataset and Basic Concepts

As stated in Section 1, we had access to four documents from a railway PCS, each one (P1, P2, P3, P4) of them with a structure consisting in three main features: object identifiers, DSs and (optional) CRs associated with these specifications. These documents follow section hierarchies, with sections such as Conventions or System Overview.

A DS is formally described as a PCS standard product feature written in NL and contains a set of sentences. In addition, it comprises various types of interfaces and activation functions which are some of the components that compose a product feature, as shown in Figure 3.2a.

A CR consists of a revision of a product feature to adapt the product feature to the needs of a specific customer of the system. It complements the standard feature description integrated into the component specification with new information defined by a customer. A CR is also written in NL and identified with a unique code. Figure 3.3 shows a CR associated with the DS of Figure 3.2a.
REQ-290: If the BGW signal is not valid the vehicle speed shall locally be calculated based on the axle speed signals from the Converter.

During braking, coasting and during start of the train the local vehicle speed shall be calculated as the second highest of all valid axle speed signals.

During traction the local vehicle speed shall be calculated as the second lowest of all valid axle speed signals.

The highest respectively the lowest axle speed shall be used when not all speed signals are valid.

Figure 3.3: Example of a customer requirement associated with the design specification of Figure 3.2a.

Each DS is identified by an object identifier and may be related to multiple CRs. On the other hand, customer requirements associated with the same design specification are considered independent (i.e., there are no relationships between them). These elements were not preprocessed before being handed to us. We do not state the actual name of the PCSs we work with and the complete object identifiers throughout this document for intellectual property reasons.

We performed exploratory data analysis on all the subsystems, and we realized that each one followed a different structure, as described in Table 3.2. To face these different structures, we had to define a parser able to handle them and standardize their formats to a single one. We chose as our standard format the one of the P1 subsystem, the most recent built subsystem within the four we handle in this dissertation. After parsing and removing non-English annotations, we obtained the subsystems in the standard format. Table 3.1, columns 2–4, shows the number of object identifiers, DSs and CRs each subsystem contains after this step.

3.3 The ARRINA Model

In this section, we present the model we defined to provide recommendations that will help engineers reduce the time spent in associating requirements and testing costs. First, we introduce the Preprocessing task, which follows an IE pipeline to tokenize and tag all the sentences from the inputs: the DSs and CRs of subsystems and the input requirements we want to provide recommendations to. These tagged sentences are necessary for the task following: the Main Chunking task, divided into noun phrase chunking and frequency refinement processes to extract the relevant components of each domain. Based on these relevant components, the Rule and Knowledge Representation task takes place to provide the knowledge graph and the set of association rules of each domain in the subsystem. Lastly, the Recommendation task provides the top-k recommendations for each input requirement. Figure 3.4 gives a high-level overview of the model with these tasks, each one detailed in the following subsections.
Table 3.2: Subsystem descriptions.

<table>
<thead>
<tr>
<th>PCS</th>
<th>Description</th>
</tr>
</thead>
</table>
| **P1** | • All rows contain an object identifier.  
• Specifications and requirements are described in natural language. |
| **P2** | • Rows with object identifier refer to specifications. Some rows do not have an object identifier - descriptions do not refer to proper specifications.  
• Specifications are described in natural language.  
• Requirements are described in natural language and include annotations or comments in non-English language. |
| **P3** | • Rows with object identifier refer to specifications. Some rows do not have an object identifier - descriptions do not refer to proper specifications.  
• Specifications are described in natural language.  
• Existence of an extra column with redundant information referring to design specifications.  
• Requirements are described as a dictionary with various features, natural language description included. |
| **P4** | • All rows contain an object identifier.  
• Specifications are described in natural language.  
• Requirements are described in natural language, and include annotations in non-English language and pseudocode. |

### 3.3.1 Preprocessing

This phase is focused in formatting the NL sentences from *design specifications*, *customer requirements* and *input requirements* in *noun phrase chunks*, the last being the one of the main inputs to association rule extraction. This phase is divided in four steps:

1. **Non-Specification Identification.** This step aims to identify contents not considered specifications, do not have any associated requirements, and do not bring any value to the analysis. We initially thought that this identification should be understood as a manual, brute-force process. However, this is unfeasible if we are working with subsystems with a high number of *specifications*. The following contents are identified and discarded for further evaluation:

   • section, subsection and category titles;  
   • comments (lowered-case text starting with "comment:'");  
   • lowered-case text starting with the substrings "refer to chapter’ or "function overview";  
   • text with less than seven tokens [6].
2. **Tokenization.** The whole string corresponding to text is split into *tokens*. A token consists of a word or punctuation character. Since we work with text from technical sources, splitting problems have not arisen (e.g., recognize apostrophes as words). Hence, the *tokenizer* splits sentences into distinct tokens.

3. **Stopwords Removal.** Tokens corresponding to stopwords (e.g., “a”, “the”, “us”) are removed. These do not contribute to the efficiency of the model and do not add value to IR tasks.

4. **POS Tagging.** For each sentence of a DS, a CR, or an input requirement, each token is tagged with its specific POS tag, using the Averaged Perceptron Model [67] with the Penn Treebank Tagset [1]. For example, Figure 3.2b is the result of this step when applied to the DS of Figure 3.2a: these tags are used as input to the next phase of the pipeline, the *Main Chunking* pipeline, described next.

### 3.3.2 Main Chunking

The *main chunking* phase receives a set of DSs, CRs and input requirements, with their sentences already tagged, performs the Noun Phrase Chunking task over each sentence, and then refines the results to obtain the most relevant chunks, the *main chunks* (defined in Section 3.3.2.2).
3.3.2.1 Noun Phrase Chunking

In order to recognize and extract noun phrase chunks (chunks for short), i.e., subsystem components, we created a chunk grammar (described in Table 3.3) with four rules (expressed as regular expressions) that define the sequence of tags to be discovered and that can be associated with the presence of components.

Table 3.3: The rules defined in the chunk grammar.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>{&lt;JJ&gt;+&lt;NNP&gt;+}</td>
<td>At least one adjective (JJ) followed by at least by a proper noun (NNP)</td>
</tr>
<tr>
<td>{&lt;NN&gt;<em>&lt;NNP&gt;+&lt;NN&gt;</em>}</td>
<td>An noun (NN) (if exists) followed by at least a proper noun (NNP), followed by another noun (NN) (if exists)</td>
</tr>
<tr>
<td>{&lt;JJ&gt;+&lt;NN&gt;}</td>
<td>At least one adjective (JJ) followed by at least a noun (NN)</td>
</tr>
<tr>
<td>{&lt;NN&gt;&lt;NN&gt;}</td>
<td>Two consecutive nouns (NN)</td>
</tr>
</tbody>
</table>

A tagged sentence is processed from left to right, rules are applied from top to bottom, as explained in Section 2.3. This implies that patterns are employed in order, and the first rule that discovers a chunk in a subset of tokens is the one used.

3.3.2.2 Main Chunking Refinement

After obtaining the chunks and making a manual analysis over them, we realized that several chunks derived from each other. A chunk $A$ derives from a chunk $B$ if $B$ is a substring of $A$, and $B$ occurs more frequently than $A$. To get the most relevant components and precise results, the main chunk concept was introduced, which denotes a chunk that is a substring of another chunk (i.e., chunk $B$), and that has a frequency above a threshold $t$. Therefore, after processing the tagged sentences, the main chunking refinement task processes the whole set of chunks into a manually defined threshold $t$ to extract the main chunks. Also, the set of derived chunks of each main chunk is obtained. Figure 3.5 depicts the refinement of two main chunks - $AC$ and $MC$, in sets of derived chunks. We do not state the actual name of some of the names of the chunks and the main chunks we extract throughout this document for intellectual property reasons.

![Figure 3.5: Simple depiction of main chunks and their derived chunks.](image-url)
3.3.3 Rule and Knowledge Representation

The Rule and Knowledge Representation phase receives as input the main chunks and their derived chunks of a subsystem (i.e., the DSs and CRs that have been processed) and outputs the association rules discovered between the main chunks and the knowledge graph representative of them. The phase employs a Weight Association Rule Mining (WARM) process, which evolves the standard ARM methodology with the weight of each chunk that occurs in the dataset.

The WARM process is boosted by the Weight Attribution and Frequent Itemset Extraction tasks that, respectively, calculate the weight of participation for each chunk and extract itemsets where they occur within the dataset. Afterwards, the rules are extracted by WARM.

3.3.3.1 Weight Attribution

We look to define weights for each main chunk. For that, first, each chunk \( c \) (either a main chunk \( mc \) or a derived chunk \( dc \)) will have an associated uniqueness value (UV), which depends on the number of main chunks from where it derives, as described in Equation 3.1.

\[
UV(c) = \frac{1}{\#(related \ main \ chunks)_c}
\]  

(3.1)

Secondly, the weight \( W \) of each main chunk \( mc \) is determined as being the sum of the UV values of its derived chunks, as described in Equation 3.2.

\[
W(mc) = \sum_{i=1}^{N} UV(dc_i)
\]  

(3.2)

In order to avoid a great discrepancy between values, we apply min-max normalization \([59] \) to \([0,1]\) range over the resulting weight array of main chunks. Since Equation 3.3 uses a product of the \( W(mc) \) and so to avoid null results, we perform value smoothing by adding the thousandth unit to the null weights to circumvent giving zero-value for them. Similarly to what we do with null weights, we also apply smoothing over unit weights by subtracting the thousandth unit as well in order to keep uniformity in the task. Therefore, we will not have weights equalling zero or one.

3.3.3.2 Frequent Itemset Extraction

A binary occurrence dataset is created. This dataset sets the association between a DS/CR, encoded as transaction and the main chunks that occur within.

Transactions are obtained following the reasoning described in Section 2.4. Given a transaction \( T \) and a main chunk \( mc \): if \( mc \in T \), \( T[mc] = 1 \), otherwise \( T[mc] = 0 \). For example, if we have 15 main chunks and a specification \( S \) contains five main chunks, an instance of the binary dataset will be composed of 15 elements, where five of them will have their values equal to 1. Afterwards, further analysis is made to recognize if some specifications or requirements do not contain any main chunk, i.e., instances of the
dataset with all their values equal to 0. These are considered irrelevant and are removed since they do not add any relevant information.

After this refinement, the model performs the frequent itemset extraction task over the binary occurrence dataset, invoking the Apriori [7] algorithm to extract itemsets with support above a manually defined threshold. These itemsets are required to build the association rules that express relations between components of DSs or CRs.

### 3.3.3.3 Weight Association Rule Mining (WARM)

We defined a new ARM framework better to use the derived chunks of each main chunk. This is described in the following pipeline:

For each itemset \{mc_1, ..., mc_n\} (or it for short) resulting from the previous task, it is defined its Itemset Transaction Weight (ITW) (represented in Equation 3.3) that reflects the aggregated weight of all main chunks it contains.

\[
ITW\{mc_1, ..., mc_n\} = \prod_{i=1}^{n} W(mc_i)
\]  

(3.3)

However, the metrics of support, confidence, and lift, described in Section 2.4, needed to be adjusted for ITW. For an itemset it, weighted support (WS) is calculated through Equation 3.4. We also made adjustments in the number of transactions, since they depend directly on the WS.

\[
WS(it) = \frac{T_{it} \times ITW(it)}{\#(transactions)}
\]

(3.4)

Representing an association rule by the general \(A \Rightarrow B\) form, where \(A = \{mc_1, ..., mc_n\}\) and \(B = \{mc_m\}\), and being \(A \Rightarrow B\) defined by an antecedent A and a consequent B, such as \(A \cap B = \emptyset\), weighted confidence (WC) and weighted lift (WL) are obtained through Equations 3.5 and 3.6.

\[
WC(A \Rightarrow B) = \frac{WS(A \cup B)}{WS(A)}
\]

(3.5)

\[
WL(A \Rightarrow B) = \frac{WS(A \cup B)}{WS(A) \times WS(B)}
\]

(3.6)

The extracted rules follow a minimum WC threshold and are ordered first by WC values. In the case of a tie, rules are ordered by WS and then by WL. The knowledge graph that represents the relations between components (i.e., main chunks) is also created, where a graph node corresponds to a main chunk that occurs in the set of rules, including notation with its amount of derived chunks and a directional edge corresponds to a rule between two main chunks. As two main chunks might occur in several rules together, the highest possible confidence is represented in the edge for combining the two main chunks with colouring methods.
3.3.4 Recommendation

The Recommendation task is focused on providing top-k DSs as recommendations for a set of input requirements through similarity techniques between the chunks (both main and derived) contained in the specifications and input requirements. This task requires:

- a subsystem, with the design specifications and their extracted main and derived chunks;
- one or more input requirements (either CRs or main transformer requirements), with the derived chunks and the main chunks referring to the domain of the subsystem to be targeted.

An input requirement is compared with all the existing design specifications, and the model will output which are the K most similar specifications and their respective CRs linked. Similarity between a design specification and a customer requirement is computed following the IoCwgt metric described by Azevedo et al. [9] and in Equations 2.15 and 3.8. The case with the metric is that it can provide a better sense in terms of the participation of chunks than standard metrics such as the Jaccard Index. We use the chunks as the inputs for the metric, though we also did some experimentation with the main chunks to understand which type of chunking framework would provide the most accurate results.

Figure 3.6 depicts the computation of the similarity between a design specification and a customer requirement: the design specification has 6 chunks (|ds|) in its description, while the customer requirement has 2 chunks (|cr|). Their intersection (|ds ∩ cr|) consists in one chunk: [COMPONENT], identified in green in Figure 3.6. For reference, any chunk a and any chunk b are similar if a can be found as a substring of b or if b is a substring of a. While the Jaccard Index would compute a similarity value of 0, 143, as described in Equation 3.7, the IoCwgt retrieves a similarity value of 0, 333 (given by Equation 3.8) that is more relevant given the set sizes of the DSs and the customer requirement involved in the computation. In other words, for our recommendation system is more relevant to measure how much each chunk participates in the link between a DS and a input requirement than the textual similarity of these. Our model does not filter results with minimum similarity thresholds, although with other domains it might be possible to perform filtering.

```plaintext
REQ-071: Loss of forced cooling (e.g. loose impeller) shall be detected (e.g. cos-fi relay, heat sink temperature supervision), to reduce the risk for overheating of [COMPONENT] installed without ambient air temperature supervision.

SSHA80: The cooling of [COMPONENT] shall be assured by supervision and/or type testing.
```

Figure 3.6: Similarity between a design specification (top) and a customer requirement (bottom).
\[ \text{sim}_{\text{Jaccard}}(ds, cr) = \frac{|ds \cap cr|}{|ds \cup cr|} = \frac{|ds \cap cr|}{|ds| + |cr| - |ds \cap cr|} \]
\[ = \frac{1}{6 + 2 - 1} = 0.143 \quad (3.7) \]

\[ \text{IoC}_{\text{wgt}}(ds, cr) = \frac{|ds \cap cr| \times (|ds| + |cr|)}{2 \times |ds| \times |cr|} \]
\[ = \frac{1 \times (6 + 2)}{2 \times 6 \times 2} = 0.333 \quad (3.8) \]

Top-K recommendations (Figure 3.7) are provided as it follows: for an input requirement \( ir_i \) and a set of specifications \( S \), we compute the similarity with any \( s_j \in S \). The \( K \) specifications with the highest similarity values are then retrieved, with their already associated requirements.

![Diagram of Top-K recommendations](image.png)

Figure 3.7: Top-K recommendations for an input requirement.

However, before reaching the definitive pipeline for the proposed approach, we experimented with different techniques to better understand which approach could better suit the problem we are working with. In Section 3.3.4.1 we tried to solve the STS problem with Transformers, while in Section 3.3.4.2 we explored similarity with \( \text{IoC}_{\text{wgt}} \) and validated it with regression methods.

### 3.3.4.1 Similarity with Transformers

The Transformer architecture currently provides state-of-the-art performance in NLP tasks, such as STS. In our experiments, we resorted to SBERT [57], which architecture can be visualized in Figure 3.8 when it produces the sentence embeddings necessary to perform similarity. The sentences \( A \) and \( B \) are fed into a pre-trained model, followed by a pooling operation to generate the \( u \) and \( v \) embedding vectors, and then compared through the cosine similarity metric.

SBERT allows for the use of pre-trained models and \(<\text{sentence1, sentence2, label}>\) triples as inputs that feed the model. Pre-trained models can be beneficial since they were already trained on
large datasets, and the model can be trained over these architectures. Generally, the input triples require empirical evaluation, where the label corresponds to the similarity value, between 0 and 1, defined by an expert. This is achievable in more generic domains, in part because everyone can evaluate generic sentences. However, it becomes harder to retrieve samples evaluated by experts with specific domains, mainly because this is a time-consuming process.

In our experimentation, performed along with the four subsystems, each sample represents a triple composed of \(<\text{DS, CR, match}>\) into a subsystem. The \texttt{match} attribute is calculated by the Jaccard Index, and we defined these samples as \textit{weak samples}, replacing thus the possible evaluation performed by experts. From the four subsystems, we obtained 33516 \textit{weak samples}.

Afterwards, we fitted multiple hyperparameters and pre-trained model configurations, shown in Table 3.4. The following hyperparameters were used:

- \textit{Batch Size}: number of samples used in one propagation inside the model. The training set is divided into multiple batches;

- \textit{Embedding Length}: maximal length of the sequences (in bytes) provided to the model. Longer inputs are truncated;

- \textit{Learning Rate}: learning rate of the optimizer provides how much the model changes in response to the error estimated at each step;

- \(\varepsilon\): hyperparameter to prevent any divisions by zero in implementation;

- \textit{Epochs}: number of epochs in the model;
• **Warmup Steps**: during these steps, the *learning rate* gradually increases until it reaches its maximal value. Then, it decreases linearly to zero;

• **Evaluation Steps**: after each number of these steps, the model is evaluated;

• **Optimizer**: algorithm responsible for changing the weights and learning rate to reduce the error function.

<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-Trained Model</td>
<td>STSb-RoBERTa-LARGE</td>
</tr>
<tr>
<td></td>
<td>STSb-RoBERTa-BASE</td>
</tr>
<tr>
<td></td>
<td>STSb-BERT-LARGE</td>
</tr>
<tr>
<td></td>
<td>STSb-BERT-BASE</td>
</tr>
<tr>
<td>Batch Size</td>
<td>4, 8, 16</td>
</tr>
<tr>
<td>Embedding Length</td>
<td>128</td>
</tr>
<tr>
<td>Learning Rate</td>
<td>$1e^{-5}$, $1e^{-6}$</td>
</tr>
<tr>
<td>$\varepsilon$</td>
<td>$1e^{-5}$, $2e^{-5}$</td>
</tr>
<tr>
<td>Epochs</td>
<td>1</td>
</tr>
<tr>
<td>Warmup Steps</td>
<td>10% of training set</td>
</tr>
<tr>
<td>Evaluation Steps</td>
<td>20% of training set</td>
</tr>
<tr>
<td>Optimizer</td>
<td>AdamW</td>
</tr>
</tbody>
</table>

Table 3.4: Hyperparameters tested in Transformer training.

For each hyperparameter combination, we performed five runs. We evaluated the performance by comparing the produced embeddings with the ones existing in the STS 2015 dataset (3000 triples) [10] by calculating the Spearman and Pearson rank correlation in comparison to the gold standard labels. Results are described in detail in Tables 3.5, 3.6, and 3.7. Due to time constraints and heavy computation (each epoch ran for 6 to 8 hours) required by this architecture, it was impossible to evaluate with more combinations, and configurations were run in one epoch.

The best configuration retrieved was the following - **STSb-BERT-BASE**, **embedding length of 128**, **learning rate** $= 2e^{-5}$, $\varepsilon = 1e^{-6}$. The optimizer used was AdamW [42], which implements stochastic optimization methods to modify weights in learning, and is currently one the most common optimizers used. Next, we performed similarity between *design specifications* and *Main Transformer requirements* to understand if the results obtained against gold standard labels would translate well in a more domain-specific task.

Unfortunately, the correlation with the gold standard labels did not translate well when defining the sentence embeddings of *design specifications* and comparing them with the ones produced from the *Main Transformer requirements*. After evaluation and discussion with system engineers, there was dissatisfaction about the recommendations provided, many were inaccurate.
### Table 3.5: Transformer results for batch size = 4.

<table>
<thead>
<tr>
<th>Pre-Trained Model</th>
<th>$\varepsilon$</th>
<th>$lr$</th>
<th>$R_1$</th>
<th>$R_2$</th>
<th>$R_3$</th>
<th>$R_4$</th>
<th>$R_5$</th>
<th>$\overline{R}$</th>
<th>$\sigma_R$</th>
</tr>
</thead>
<tbody>
<tr>
<td>STSB-BERT-BASE</td>
<td>$1e^{-6}$</td>
<td>$2e^{-5}$</td>
<td>0.8476</td>
<td>0.8449</td>
<td>0.841</td>
<td>0.8458</td>
<td>0.8623</td>
<td><strong>0.8483</strong></td>
<td><strong>0.0073</strong></td>
</tr>
<tr>
<td>STSB-BERT-BASE</td>
<td>$1e^{-6}$</td>
<td>$1e^{-5}$</td>
<td>0.8303</td>
<td>0.8272</td>
<td>0.8173</td>
<td>0.8186</td>
<td>0.8405</td>
<td>0.8268</td>
<td>0.0085</td>
</tr>
<tr>
<td>STSB-BERT-LARGE</td>
<td>$1e^{-6}$</td>
<td>$2e^{-5}$</td>
<td>0.8009</td>
<td>0.8414</td>
<td>0.8059</td>
<td>0.7946</td>
<td>0.7996</td>
<td>0.8085</td>
<td>0.0169</td>
</tr>
<tr>
<td>STSB-RoBERTa-BASE</td>
<td>$1e^{-6}$</td>
<td>$2e^{-5}$</td>
<td>0.8155</td>
<td>0.8143</td>
<td>0.8096</td>
<td>0.7832</td>
<td>0.8121</td>
<td>0.8069</td>
<td>0.0120</td>
</tr>
<tr>
<td>STSB-BERT-BASE</td>
<td>$1e^{-5}$</td>
<td>$2e^{-5}$</td>
<td>0.8056</td>
<td>0.7907</td>
<td>0.7859</td>
<td>0.7856</td>
<td>0.7955</td>
<td>0.7926</td>
<td>0.0074</td>
</tr>
<tr>
<td>STSB-BERT-BASE</td>
<td>$1e^{-5}$</td>
<td>$1e^{-5}$</td>
<td>0.7973</td>
<td>0.7866</td>
<td>0.7845</td>
<td>0.7766</td>
<td>0.7892</td>
<td>0.7868</td>
<td>0.0067</td>
</tr>
<tr>
<td>STSB-RoBERTa-BASE</td>
<td>$1e^{-6}$</td>
<td>$1e^{-5}$</td>
<td>0.7668</td>
<td>0.7956</td>
<td>0.7918</td>
<td>0.7582</td>
<td>0.7795</td>
<td>0.7784</td>
<td>0.0143</td>
</tr>
<tr>
<td>STSB-BERT-LARGE</td>
<td>$1e^{-6}$</td>
<td>$1e^{-5}$</td>
<td>0.7133</td>
<td>0.7720</td>
<td>0.7722</td>
<td>0.7308</td>
<td>0.7765</td>
<td>0.7530</td>
<td>0.0259</td>
</tr>
<tr>
<td>STSB-RoBERTa-BASE</td>
<td>$1e^{-5}$</td>
<td>$2e^{-5}$</td>
<td>0.7555</td>
<td>0.7587</td>
<td>0.7614</td>
<td>0.6985</td>
<td>0.7064</td>
<td>0.7361</td>
<td>0.0276</td>
</tr>
<tr>
<td>STSB-RoBERTa-BASE</td>
<td>$1e^{-5}$</td>
<td>$1e^{-5}$</td>
<td>0.7354</td>
<td>0.7541</td>
<td>0.7591</td>
<td>0.7049</td>
<td>0.6940</td>
<td>0.7295</td>
<td>0.0260</td>
</tr>
<tr>
<td>STSB-BERT-LARGE</td>
<td>$1e^{-5}$</td>
<td>$1e^{-5}$</td>
<td>0.7547</td>
<td>0.7199</td>
<td>0.7579</td>
<td>0.7108</td>
<td>0.7002</td>
<td>0.7287</td>
<td>0.0234</td>
</tr>
<tr>
<td>STSB-BERT-LARGE</td>
<td>$1e^{-5}$</td>
<td>$2e^{-5}$</td>
<td>0.7216</td>
<td>0.7501</td>
<td>0.7536</td>
<td>0.6597</td>
<td>0.7025</td>
<td>0.7175</td>
<td>0.0345</td>
</tr>
<tr>
<td>STSB-RoBERTa-LARGE</td>
<td>$1e^{-6}$</td>
<td>$2e^{-5}$</td>
<td>0.5330</td>
<td>0.5544</td>
<td>0.7454</td>
<td>0.7289</td>
<td>0.5845</td>
<td>0.6292</td>
<td>0.0898</td>
</tr>
<tr>
<td>STSB-RoBERTa-LARGE</td>
<td>$1e^{-6}$</td>
<td>$1e^{-5}$</td>
<td>0.4587</td>
<td>0.4853</td>
<td>0.6780</td>
<td>0.6416</td>
<td>0.5236</td>
<td>0.5575</td>
<td>0.0868</td>
</tr>
<tr>
<td>STSB-RoBERTa-LARGE</td>
<td>$1e^{-5}$</td>
<td>$1e^{-5}$</td>
<td>0.4918</td>
<td>0.5459</td>
<td>0.5316</td>
<td>0.5996</td>
<td>0.4514</td>
<td>0.5241</td>
<td>0.0501</td>
</tr>
<tr>
<td>STSB-RoBERTa-LARGE</td>
<td>$1e^{-5}$</td>
<td>$2e^{-5}$</td>
<td>0.4665</td>
<td>0.4855</td>
<td>0.5520</td>
<td>0.6302</td>
<td>0.4785</td>
<td>0.5225</td>
<td>0.0615</td>
</tr>
</tbody>
</table>

$\varepsilon$: Learning Rate  $lr$: Model Run  $\overline{R}$: Average  $\sigma_R$: Standard Deviation

### Table 3.6: Transformer results for batch size = 8.

<table>
<thead>
<tr>
<th>Pre-Trained Model</th>
<th>$\varepsilon$</th>
<th>$lr$</th>
<th>$R_1$</th>
<th>$R_2$</th>
<th>$R_3$</th>
<th>$R_4$</th>
<th>$R_5$</th>
<th>$\overline{R}$</th>
<th>$\sigma_R$</th>
</tr>
</thead>
<tbody>
<tr>
<td>STSB-BERT-BASE</td>
<td>$1e^{-6}$</td>
<td>$2e^{-3}$</td>
<td>0.8526</td>
<td>0.8512</td>
<td>0.8454</td>
<td>0.8700</td>
<td>0.8553</td>
<td><strong>0.8549</strong></td>
<td><strong>0.0082</strong></td>
</tr>
<tr>
<td>STSB-BERT-BASE</td>
<td>$1e^{-6}$</td>
<td>$1e^{-3}$</td>
<td>0.8269</td>
<td>0.8233</td>
<td>0.8204</td>
<td>0.8433</td>
<td>0.8384</td>
<td>0.8323</td>
<td>0.0074</td>
</tr>
<tr>
<td>STSB-RoBERTa-BASE</td>
<td>$1e^{-6}$</td>
<td>$2e^{-5}$</td>
<td>0.8360</td>
<td>0.8164</td>
<td>0.8226</td>
<td>0.8158</td>
<td>0.8202</td>
<td>0.8222</td>
<td>0.0073</td>
</tr>
<tr>
<td>STSB-BERT-BASE</td>
<td>$1e^{-5}$</td>
<td>$2e^{-5}$</td>
<td>0.8119</td>
<td>0.7963</td>
<td>0.7931</td>
<td>0.8115</td>
<td>0.8236</td>
<td>0.8073</td>
<td>0.0112</td>
</tr>
<tr>
<td>STSB-BERT-BASE</td>
<td>$1e^{-5}$</td>
<td>$1e^{-5}$</td>
<td>0.7966</td>
<td>0.7897</td>
<td>0.7976</td>
<td>0.8124</td>
<td>0.8180</td>
<td>0.8029</td>
<td>0.0106</td>
</tr>
<tr>
<td>STSB-RoBERTa-BASE</td>
<td>$1e^{-6}$</td>
<td>$1e^{-5}$</td>
<td>0.8230</td>
<td>0.7757</td>
<td>0.8037</td>
<td>0.8049</td>
<td>0.8058</td>
<td>0.8026</td>
<td>0.0152</td>
</tr>
<tr>
<td>STSB-BERT-BASE</td>
<td>$1e^{-5}$</td>
<td>$2e^{-5}$</td>
<td>0.7932</td>
<td>0.7369</td>
<td>0.7711</td>
<td>0.7946</td>
<td>0.7996</td>
<td>0.7926</td>
<td>0.0074</td>
</tr>
<tr>
<td>STSB-RoBERTa-BASE</td>
<td>$1e^{-5}$</td>
<td>$1e^{-5}$</td>
<td>0.7820</td>
<td>0.7185</td>
<td>0.7844</td>
<td>0.7856</td>
<td>0.7955</td>
<td>0.7868</td>
<td>0.0067</td>
</tr>
<tr>
<td>STSB-RoBERTa-LARGE</td>
<td>$1e^{-6}$</td>
<td>$2e^{-5}$</td>
<td>0.6993</td>
<td>0.7786</td>
<td>0.7949</td>
<td>0.7582</td>
<td>0.7567</td>
<td>0.6209</td>
<td>0.0715</td>
</tr>
<tr>
<td>STSB-RoBERTa-LARGE</td>
<td>$1e^{-6}$</td>
<td>$1e^{-5}$</td>
<td>0.7956</td>
<td>0.7185</td>
<td>0.7844</td>
<td>0.7856</td>
<td>0.7955</td>
<td>0.7926</td>
<td>0.0074</td>
</tr>
<tr>
<td>STSB-RoBERTa-LARGE</td>
<td>$1e^{-5}$</td>
<td>$2e^{-5}$</td>
<td>0.5539</td>
<td>0.6676</td>
<td>0.6570</td>
<td>0.7229</td>
<td>0.5958</td>
<td>0.6394</td>
<td>0.0588</td>
</tr>
<tr>
<td>STSB-RoBERTa-LARGE</td>
<td>$1e^{-5}$</td>
<td>$1e^{-5}$</td>
<td>0.5239</td>
<td>0.6836</td>
<td>0.6592</td>
<td>0.6807</td>
<td>0.5485</td>
<td>0.6192</td>
<td>0.0687</td>
</tr>
</tbody>
</table>

$\varepsilon$: Learning Rate  $lr$: Model Run  $\overline{R}$: Average  $\sigma_R$: Standard Deviation
Chapter 3 ARRINA System

<table>
<thead>
<tr>
<th>Pre-Trained Model</th>
<th>$\varepsilon$</th>
<th>$lr$</th>
<th>$R_1$</th>
<th>$R_2$</th>
<th>$R_3$</th>
<th>$R_4$</th>
<th>$R_5$</th>
<th>$T$</th>
<th>$\sigma_T$</th>
</tr>
</thead>
<tbody>
<tr>
<td>STSb-BERT-BASE</td>
<td>$1e^{-6}$</td>
<td>$2e^{-5}$</td>
<td>0.8900</td>
<td>0.8765</td>
<td>0.8598</td>
<td>0.8826</td>
<td>0.8895</td>
<td>0.8797</td>
<td>0.0111</td>
</tr>
<tr>
<td>STSb-BERT-BASE</td>
<td>$1e^{-6}$</td>
<td>$1e^{-5}$</td>
<td>0.8785</td>
<td>0.8607</td>
<td>0.8356</td>
<td>0.8639</td>
<td>0.8741</td>
<td>0.8626</td>
<td>0.0149</td>
</tr>
<tr>
<td>STSb-BERT-BASE</td>
<td>$1e^{-5}$</td>
<td>$2e^{-5}$</td>
<td>0.8611</td>
<td>0.8294</td>
<td>0.8148</td>
<td>0.8501</td>
<td>0.8580</td>
<td>0.8427</td>
<td>0.0178</td>
</tr>
<tr>
<td>STSb-BERT-BASE</td>
<td>$1e^{-5}$</td>
<td>$1e^{-5}$</td>
<td>0.8540</td>
<td>0.8233</td>
<td>0.8019</td>
<td>0.8445</td>
<td>0.8525</td>
<td>0.8353</td>
<td>0.0199</td>
</tr>
<tr>
<td>STSb-RoBERTa-BASE</td>
<td>$1e^{-6}$</td>
<td>$2e^{-5}$</td>
<td>0.8384</td>
<td>0.8384</td>
<td>0.8384</td>
<td>0.8384</td>
<td>0.8384</td>
<td>0.8355</td>
<td>0.0046</td>
</tr>
<tr>
<td>STSb-BERT-LARGE</td>
<td>$1e^{-6}$</td>
<td>$2e^{-5}$</td>
<td>0.8494</td>
<td>0.8567</td>
<td>0.8589</td>
<td>0.6983</td>
<td>0.8677</td>
<td>0.8262</td>
<td>0.0642</td>
</tr>
<tr>
<td>STSb-BERT-LARGE</td>
<td>$1e^{-6}$</td>
<td>$1e^{-5}$</td>
<td>0.8162</td>
<td>0.8276</td>
<td>0.8194</td>
<td>0.8231</td>
<td>0.8191</td>
<td>0.8211</td>
<td>0.0039</td>
</tr>
<tr>
<td>STSb-RoBERTa-BASE</td>
<td>$1e^{-5}$</td>
<td>$2e^{-5}$</td>
<td>0.8056</td>
<td>0.8101</td>
<td>0.8146</td>
<td>0.8009</td>
<td>0.7993</td>
<td>0.8061</td>
<td>0.0057</td>
</tr>
<tr>
<td>STSb-RoBERTa-BASE</td>
<td>$1e^{-5}$</td>
<td>$1e^{-5}$</td>
<td>0.8255</td>
<td>0.8170</td>
<td>0.8437</td>
<td>0.7190</td>
<td>0.8150</td>
<td>0.8040</td>
<td>0.0437</td>
</tr>
<tr>
<td>STSb-RoBERTa-BASE</td>
<td>$1e^{-5}$</td>
<td>$1e^{-5}$</td>
<td>0.7813</td>
<td>0.7890</td>
<td>0.8095</td>
<td>0.7920</td>
<td>0.7957</td>
<td>0.7935</td>
<td>0.0093</td>
</tr>
<tr>
<td>STSb-RoBERTa-BASE</td>
<td>$1e^{-5}$</td>
<td>$1e^{-5}$</td>
<td>0.7955</td>
<td>0.8210</td>
<td>0.8147</td>
<td>0.7547</td>
<td>0.7776</td>
<td>0.7927</td>
<td>0.0244</td>
</tr>
<tr>
<td>STSb-RoBERTa-LARGE</td>
<td>$1e^{-6}$</td>
<td>$1e^{-5}$</td>
<td>0.7873</td>
<td>0.7940</td>
<td>0.7699</td>
<td>0.7884</td>
<td>0.7681</td>
<td>0.7820</td>
<td>0.0108</td>
</tr>
<tr>
<td>STSb-RoBERTa-LARGE</td>
<td>$1e^{-5}$</td>
<td>$2e^{-5}$</td>
<td>0.7489</td>
<td>0.7486</td>
<td>0.7375</td>
<td>0.7107</td>
<td>0.7281</td>
<td>0.7348</td>
<td>0.0143</td>
</tr>
<tr>
<td>STSb-RoBERTa-LARGE</td>
<td>$1e^{-6}$</td>
<td>$2e^{-5}$</td>
<td>0.8161</td>
<td>0.3041</td>
<td>0.8366</td>
<td>0.8188</td>
<td>0.7904</td>
<td>0.7223</td>
<td>0.2104</td>
</tr>
<tr>
<td>STSb-RoBERTa-LARGE</td>
<td>$1e^{-5}$</td>
<td>$1e^{-5}$</td>
<td>0.7067</td>
<td>0.7370</td>
<td>0.7408</td>
<td>0.6824</td>
<td>0.7302</td>
<td>0.7194</td>
<td>0.0220</td>
</tr>
</tbody>
</table>

$\varepsilon$: Learning Rate  $lr$: Model Run  $T$: Average  $\sigma_T$: Standard Deviation

Table 3.7: Transformer results for batch size = 16.

Many arguments can be made for the lack of accuracy: STS is complex and depends on semantic structure, not only keywords. At the same time, though we feed the model with weak samples, experts do not evaluate these and depend on terminology that cannot be found in generic corpora. Also, the training process is costly, so we decided to look for more straightforward solutions through token-based similarity techniques and, therefore, put away the approach described in this section.

3.3.4.2 Similarity Metric Validation

We assessed the metric $IoC_{wgt}$ with regression methods to understand if the metric could be learnt by just requiring information from textual features of specifications and requirements. To do so, for each subsystem, we defined a set of samples consisting of all the existing DS-CR links it contains.

Table 3.8 depicts an example of how samples are structured for three linkages $(s_i, r_j)$. Column 2 contains the specification $s_i$ identifier, while the next five columns ($MC_{spec}$) represent the set of main chunks contained in all specifications of the target subsystem. These columns are filled with binary value which denotes the presence of the main chunk $mc_i$ in $s_i$ (value 1) or not (value 0). For requirement $r_j$, column 8 depicts its identifier and columns 9–13 represent the set of main chunks ($MC_{req}$) found in requirements of the target subsystem. These columns also contain binary values and are filled the same way as $MC_{spec}$, but for requirements. The last two columns - $main_{loc}$ and $chunk_{loc}$ - refer to the similarity for $(s_i, r_j)$ with main chunks (columns 3–7 and columns 9–13) and all chunks (all extracted chunks, including the main chunks) respectively. Both similarities are calculated using $IoC_{wgt}$ metric.

Table 3.9 shows an example of a $(s_i, r_j)$ link from the $P1$ subsystem with 100 main chunks from DSs
and 66 main chunks from CRs. $s_i$ has four main chunks [AC, Motor Converter, TCMS, tractive effort] and will be represented by a vector of 100 elements with four indexes as 1 (the others as 0); $r_j$ has five main chunks [AC, Motor Converter, AUX, LC, main transformer] and will generate a vector of 66 elements with five indexes as 1 (the others as 0). From these two vectors, we calculate the main $ioc$, where 0.45 indicates there is good participation of the main chunks in the link.

Based on main $ioc$ and chunk $ioc$ values of all samples of a subsystem, we divide the samples in three sets of cases:

- Case C1 (row 1 of Table 3.8): $main_{ioc} > chunk_{ioc}$, meaning that main chunks have a stronger presence in the linking;
- Case C2 (row 2 of Table 3.8): $main_{ioc} < chunk_{ioc}$, meaning that general chunks prevail over the main chunks;
- Case C3 (row 3 of Table 3.8): $main_{ioc} = 0$, 0 and $chunk_{ioc} = 0$, 0, meaning that there is no relation between the specification $s_i$ and the requirement $r_j$.

We evaluated four regression methods, namely, Decision Tree Regression, Bayesian Ridge Regression, Support Vector Regression and Random Forest Regression, over different combinations of the sets of Case 1 to Case 3. The combinations used were: C1 - samples of Case 1; C2 - samples of Case 2; (C12) - samples of both Cases 1 and 2; (SMOGEN C12) - samples of both Cases 1 and 2, binded with sampling methods; and (FULL, consisting of the union of C1, C2, and C3). C3 was not evaluated independently, because its samples have the same value of $main_{ioc}$ and $chunk_{ioc}$, and, therefore, the results will be perfect, transmitting a false reliability to the model.

We produced a train-test split of 80%/20% for each combination and performed grid-search and cross-validation pipelines over the training set. In grid-search, we performed 5-fold splitting, with the parameters described in Table 3.10 and the performance evaluated with $R^2$ scoring. The best performing model

<table>
<thead>
<tr>
<th>Case</th>
<th>Specification</th>
<th>$MC_{spec}$</th>
<th>Requirement</th>
<th>$MC_{req}$</th>
<th>$main_{ioc}$</th>
<th>$chunk_{ioc}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>A</td>
<td>0 1 0 ... 1</td>
<td>B</td>
<td>1 0 1 ... 1</td>
<td>0.789</td>
<td>0.601</td>
</tr>
<tr>
<td>C2</td>
<td>B</td>
<td>0 0 1 ... 0</td>
<td>C</td>
<td>0 0 0 ... 0</td>
<td>0.410</td>
<td>0.532</td>
</tr>
<tr>
<td>C3</td>
<td>A</td>
<td>0 1 0 ... 1</td>
<td>C</td>
<td>0 0 0 ... 0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Table 3.8: Samples for the $IoC_{wgt}$ regression evaluation.

<table>
<thead>
<tr>
<th>Specification</th>
<th>$MC_{spec}$</th>
<th>Requirement</th>
<th>$MC_{req}$</th>
<th>$main_{ioc}$</th>
<th>$chunk_{ioc}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_i$</td>
<td>AC</td>
<td>Motor Converter</td>
<td>TCMS</td>
<td>...</td>
<td>tractive effort</td>
</tr>
</tbody>
</table>

Table 3.9: Regression evaluation for a specification-requirement pair.
is then fed into the 10-fold cross-validation pipeline: the training set is split into ten smaller sets - folds; for each of the ten folds, the model is trained in the others, and the fold is used as test data. Metrics such as $R^2$, $MAE$ and $MSE$ are computed. Figure 3.9 illustrates how the method is performed.

Table 3.10: Parameters tested for grid-search in $IoC_{wgt}$ regression.

<table>
<thead>
<tr>
<th>Regression Technique</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>DecisionTreeRegressor</td>
<td>'criterion': ('mse', 'friedman_mse', 'mae')</td>
</tr>
<tr>
<td></td>
<td>'splitter': ('best', 'random')</td>
</tr>
<tr>
<td></td>
<td>'max_depth': (None, 2, 3, 4, 5)</td>
</tr>
<tr>
<td></td>
<td>'min_samples_split': (2, 3, 4)</td>
</tr>
<tr>
<td></td>
<td>'min_samples_leaf': (2, 3, 4)</td>
</tr>
<tr>
<td>BayesianRidge</td>
<td>'alpha_1': (1e-7, 1e-6, 1e-5, 1e-4)</td>
</tr>
<tr>
<td></td>
<td>'alpha_2': (1e-7, 1e-6, 1e-5, 1e-4)</td>
</tr>
<tr>
<td></td>
<td>'lambdau1': (1e-7, 1e-6, 1e-5, 1e-4)</td>
</tr>
<tr>
<td></td>
<td>'lambdau2': (1e-7, 1e-6, 1e-5, 1e-4)</td>
</tr>
<tr>
<td>SVR</td>
<td>'C': (0.1, 1.0, 5.0, 10.0)</td>
</tr>
<tr>
<td></td>
<td>'gamma': (1, 0.1, 0.05, 0.01)</td>
</tr>
<tr>
<td></td>
<td>'kernel': ('linear', 'poly', 'rbf')</td>
</tr>
<tr>
<td></td>
<td>'epsilon': (0.1, 1.0, 5.0, 10.0)</td>
</tr>
<tr>
<td>RandomForestRegressor</td>
<td>'n_estimators': (10, 50, 200)</td>
</tr>
<tr>
<td></td>
<td>'criterion': ('mse', 'mae')</td>
</tr>
<tr>
<td></td>
<td>'max_depth': (2, 3, 4)</td>
</tr>
<tr>
<td></td>
<td>'n_jobs': (-1)</td>
</tr>
</tbody>
</table>

Figure 3.9: 10-fold cross-validation pipeline.

Through the pipeline results, described in Table 3.11, it is possible to observe that the $IoC_{wgt}$ metric can be approximated with regressors, such as Decision Tree Regressor or Support Vector Regressor, in some cases with $R^2 > 0,7$.

Though dividing the samples into three cases did not give rewarding results, it gave us a good sense of the linkages that the model did not manage to catch - the ones contained in Case 3. Also, the sampling
## Table 3.11: Regression scoring for $IoC_{wglt}$ validation.

<table>
<thead>
<tr>
<th>PCS</th>
<th>Case</th>
<th>chunk$\text{IoC}$</th>
<th>main$\text{IoC}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Samples %</td>
<td>DTR</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DTR</td>
</tr>
<tr>
<td>P1</td>
<td>C1</td>
<td>115 52.51 0.1618 -0.0541 0.1350 0.1471</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C2</td>
<td>64 29.22 -0.0029 -0.1434 -0.0770 -0.0941</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C12</td>
<td>179 81.73 0.0151 -0.0341 0.0614 0.1184</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SMOGN C12</td>
<td>797 (+36) 380.36 0.9271 0.8555 0.7423 0.5399</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FULL</td>
<td>219 100 0.0542 0.1354 0.1671 0.0886</td>
<td></td>
</tr>
<tr>
<td>P2</td>
<td>C1</td>
<td>255 56.92 0.0233 0.1760 0.1956 0.1479</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C2</td>
<td>128 28.57 0.2921 0.3619 0.3568 0.3227</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C12</td>
<td>383 85.49 0.2180 0.2919 0.3012 0.2457</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SMOGN C12</td>
<td>502 (+77) 129.24 -0.3322 -0.1218 0.0063 -0.3832</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FULL</td>
<td>448 100 0.1789 0.3188 0.3671 0.2825</td>
<td></td>
</tr>
<tr>
<td>P3</td>
<td>C1</td>
<td>17 17 -3.7299 -4.8875 -2.4333 -4.4796</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C2</td>
<td>22 22 -0.8534 -1.0339 -0.6607 -0.9748</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C12</td>
<td>39 39 0.0023 -0.1319 -0.0310 -0.0489</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SMOGN C12</td>
<td>78 (+9) 87 -0.2387 0.0529 0.3649 -0.2949</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FULL</td>
<td>100 100 0.1388 0.0347 0.3329 0.1640</td>
<td></td>
</tr>
<tr>
<td>P4</td>
<td>C1</td>
<td>34 27.20 0.2354 -0.0737 -0.2582 0.3402</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C2</td>
<td>49 39.20 0.3064 -0.1871 -0.0275 0.1152</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C12</td>
<td>83 66.40 0.6307 0.4869 0.5454 0.6164</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SMOGN C12</td>
<td>51 (+21) 57.60 0.7535 0.3981 0.7448 0.4638</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FULL</td>
<td>125 100 0.4672 0.4415 0.4665 0.4719</td>
<td></td>
</tr>
</tbody>
</table>

DTR: DecisionTreeRegressor  BR: BayesianRidge  SVR: SupportVectorRegressor  RFR: RandomForestRegressor

Methods provided by SMOGN [15] end up increasing the score in almost all the regressors, given the disparity of similarity values along with the linkages. Therefore, the metric is considered validated within the chunking framework we defined.
Chapter 4

Implementation

This chapter will go over the implementation details necessary to understand how the model was implemented, specifically its four tasks, presented in the previous chapter. First, we specify how the preprocessing pipeline was defined, followed by the main chunking framework. Then, we overview how the graph visualizations were defined and how the recommendations were created.

All tasks were developed in Python. To implement the standard NLP tasks, we resorted to the NLTK package. NLTK provides various language processing interfaces, from text processing to parsing, classification or semantic reasoning. To obtain the association rules between different components, we used MLxtend, a package that contains various data science tools. To create the graph visualizations, we employed Graphviz, an open-source visualization software also provided as a Python interface.

Figure 4.1 presents the class diagram corresponding to the modules required for the model. We decided to represent the dataset, specifications, requirements and main chunks as objects, while the other modules represent the model pipeline. In the next section, we explain each module which implements each task of the model.

4.1 Preprocessing

In this section, we look at how we can obtain DSs and CRs from an input subsystem. Given the general structure of the PCSs, consisting in \(<\text{object identifier}, \text{DS}, \text{CR}>\) triples, we need to obtain the textual descriptions of DSs and CRs to put them in the correct format.

Afterwards, it is required to clean the textual descriptions to obtain tagged sentences, which then can be used for the chunking process. Algorithm 1 describes the preprocessing methodology employed for specification or requirement processing. This algorithm requires a textual description (of a DS or a CR) to be processed, the grammar used for the noun phrase chunking and the list of stopwords, retrieving the POS tree that identifies the syntactic structure of the text. This tree already contains the token subsets identified as noun phrase chunks, defined in NLTK.
The algorithm implements the four steps of the preprocessing task. Therefore, for the text description, first it is checked if it is a non-specification content (lines 1–3). If it is the case, the algorithm ends; otherwise, it proceeds with the tokenization of the text (line 4) through the function `word_tokenize`, provided by NLTK, and removal of stopwords (lines 6–10). Finally, in lines 11–17, the POS tagging tree is generated through the grammar parser `cp`.

### 4.2 Main Chunking

To obtain the relevant chunks that will be used in the Rule and Knowledge Representation and the Recommendation tasks, we developed the main chunking framework based on frequency. Algorithm 2 shows...
Algorithm 1: Preprocessing pipeline - preprocessingPipeline().

input : Textual description text;
Grammar parser cp;
List of stopwords stops.
output: Part-of-speech tree POSTree from a textual description.
1 if nonSpecificationIdentification(text) then
2    return ""
3 end
4 tokens ← nltk.word_tokenize(text);
5 tokens_without_stopwords ← [];
6 foreach t ∈ tokens do
7        if t /∈ stops then
8            tokens_without_stopwords.add(t)
9        end
10 end
11 tags ← nltk.pos_tag(tokens_without_stopwords);
12 if tags then
13    POSTree ← parse_pos_tree(cp, tags);
14    return POSTree
15 else
16    return ""
17 end

how the set of the noun phrase chunks are extracted from a textual description and how are these joined
for frequency parsing. After the module collects all requirements and specifications of the target sub-
systems (lines 5–7), for each textual description t, it is retrieved its POSTree, by the function prepro-
cessingPipeline (line 9 - application of Algorithm 1). Next, the module iterates through each subtree of
POSTree in order to verify if their POS labels correspond to the NP tag. In such a case, we identify the
token subset as a noun phrase chunk and therefore, it is added to the list of chunks (lines 11–15). This list
is then added to an aggregator, containing chunks from every textual description (line 17). Afterwards,
we join every list of chunks to check each chunk’s frequency, returning a dictionary that matches a chunk
with its frequency and the full set of chunks in a domain (lines 19–20).

Algorithm 3 describes how the main chunks are extracted and refined. Given the dictionary obtained
in Algorithm 2, first we filter those above an integer threshold (lines 2–5). Afterwards, we compare
each possible pair of main chunks (mcA, mcB) to understand if any main chunk mcA can derive from
mcB - we look for derivation through n-grams (lines 9–23).

4.3 Rule and Knowledge Representation

This module is built by various subtasks, all depicted in Algorithm 4. The algorithm receives as input:

- The main chunks mainChunks identified in Algorithm 3;
- The whole set of chunks chunks for a specific domain;
Algorithm 2: Chunking pipeline and frequency parsing - chunkingPipeline()

\[\text{l}\]

**input**: Project requirements (specifications or requirements) \(\text{projReqs}\);
Grammar parser \(\text{cp}\);
List of stopwords \(\text{stops}\).

**output**: Set of chunks \(\text{chunks}\);
Dictionary pairing chunks and frequencies \(\text{freqs}\).

1. \(\text{chunkSet} \leftarrow ()\);
2. \(\text{chunks} \leftarrow ()\);
3. \(\text{listAggregator} \leftarrow []\);
4. \(\text{texts} \leftarrow []\);
5. \(\text{foreach} \, \text{pr} \in \text{projReqs} \, \text{do}\)
6. \(\text{texts}.\text{append}(\text{pr}.\text{text});\)
7. \(\text{end}\)
8. \(\text{foreach} \, \text{t} \in \text{texts} \, \text{do}\)
9. \(\text{POSTree} \leftarrow \text{preprocessingPipeline}(\text{t}, \text{cp}, \text{stops});\)
10. \(\text{foreach} \, \text{subtree} \in \text{POSTree}.\text{subtrees()} \, \text{do}\)
11. \(\text{if} \, \text{subtree}.\text{label()} == \text{NP} \text{then}\)
12. \(\text{npChunk} \leftarrow \text{extractChunk}(\text{subtree});\)
13. \(\text{chunkSet}.\text{add(}\text{npChunk});\)
14. \(\text{chunks}.\text{add(}\text{npChunk});\)
15. \(\text{end}\)
16. \(\text{end}\)
17. \(\text{listAggregator}.\text{append(}\text{chunkSet});\)
18. \(\text{end}\)
19. \(\text{freqs} \leftarrow \text{parseFrequency(}\text{listAggregator});\)
20. \(\text{return} \, \text{chunks} , \text{freqs};\)

• Objects referring to specifications or requirements \(\text{projReqs}\);

• Confidence threshold, an hyperparameter for rule mining \(\text{conf}\).

• Folder \(\text{folder}\) to export resulting data.

First, we define \(\text{groupingDict}\), which will hold information regarding each main chunk and its derived chunks, extracted through the function \(\text{associateDerivedChunks}\) (lines 1–5). Then, each element (DS or CR) of the target project subsystem is associated with its main chunks (lines 6–8). From the information about the main chunks and where they occur, we then calculate the weight for each main chunk (line 9). These weights will be essential for the rule mining process.

In the rule mining process (line 10–13), we need to create a binary dataframe \(\text{binaryDf}\) that defines the correspondence between each main chunk and a specification or requirement, through function \(\text{getBinaryDataset}\). The rows in \(\text{binaryDf}\) refer to a DS or a CR, while each column points to the main chunk. This dataframe will then be used to calculate the weighted frequent patterns - \(\text{getFrequentPatterns}\) - and the weighted association rules - \(\text{getAssociationRules}\) -, exported to a .xlsx file. Next, this file is used as input for the knowledge graph construction, which is achieved by the function \(\text{getKnowledgeGraph()}\) (line 14) that extracts the antecedents and the consequents of association rules as connections that constitute
Algorithm 3: Main chunk extraction and refinement - mainChunkPipeline().

```
input : Dictionary of chunks freqs;
Integer minimum threshold threshold.
output: Set of refined main chunks mainChunks.

1
mainSet ← {} // To store main chunks.
2
for ch ∈ freqs do
3    if freqs[ch] ≥ threshold then
4        mainSet.add(ch)
5    end
6 end

7 mainChunks ← mainSet.copy() // To refine main chunks.
8 mainChunks ← removeKeywords(mainChunks);
9
for i = 0 to len(mainSet) do
10    tokensA ← nltk.word_tokenize(mainSet[i].lower()); // Tokens of main chunk mcA.
11    for j = 0 to len(mainSet) do
12        if mainSet[i] ≠ mainSet[j] then
13            tokensB ← nltk.word_tokenize(mainSet[j].lower()); // Tokens of main chunk mcB.
14            nGrams ← nGrams(tokensB, len(tokensA));
15            foreach gram ∈ nGrams do
16                if gram.compare(tokensA) then
17                    mainChunks.remove(mainSet[j]);
18                    break;
19                end
20            end
21        end
22    end
23 end
24 return mainChunks;
```

the graph. The knowledge graph is exported to a .png file and stored in folder. This algorithm returns the main chunks, now containing information about their derived chunks.

4.4 Recommendation

To provide design specifications as recommendations for one or more input requirement, we have implemented Algorithm 5, which requires the following inputs:

- The input requirement inputReq;
- Target specifications specs that can be provided as recommendations;
- Type of \( IoC_{wgt} \) variation \( IoC \): the user can choose between \( main_{ioc} \) or \( chunk_{ioc} \);
- Integer threshold for top-K recommendations \( K \);
Algorithm 4: Rule and graph mining for a specific domain - representationPipeline().

```
input : List of main chunks for a specific domain mainChunks;
        List of chunks in a specific domain chunks;
        Project requirements projReqs;
        Confidence threshold conf;
        Folder folder to export data.
output: List of main chunks for a specific domain mainChunks;
1 groupingDict ← dict();
2 foreach mc ∈ mainChunks do
3    mc.derivedChunks = mc.associateDerivedChunks(mc, chunks);
4    groupingDict[mc] ← mc.derivedChunks;
5 end
6 foreach el ∈ projReqs do
7    el.mainChunks = el.setMainChunks(mainChunks)
8 end
9 weights ← defineWeights(groupingDict, chunks);
10 binaryDf ← getBinaryDataset(chunks, groupingDict, projReqs);
11 nTransactions ← len(binaryDf.index);
12 frequentPatterns ← getFrequentPatterns(nTransactions, weights, binaryDf);
13 associationRules ← getAssociationRules(nTransactions, frequentPatterns, folder) // Exports association rules to a file.
14 getKnowledgeGraph(conf, associationRules, groupingDict, folder) // Exports knowledge graph to a file.
15 return mainChunks;
```

- Type of format for the output recommendations format - the algorithm supports .xlsx and .json.

It starts by building a dictionary iocDict that will preserve the computed similarity values between inputReq and all the specifications in specs. Next, for each input requirement in evaluation, depending on the IoC variation chosen, and for each specification, the similarity value is computed. For computation, functions defineIntersection and calculateIoC are used, where the former defines the intersection set between a specification and the input requirement in analysis, while the latter calculates the similarity between them (lines 2–10). After calculation, iocDict is filtered to the K highest similarity values, in function reduceTopK (line 12). Then, this dictionary is altered to suit the format chosen in parameter format, through function buildRecommendations (line 13). This algorithm will then return the K recommendations (line 15).

### 4.5 Main Algorithm

The main algorithm (Algorithm 6) is responsible for calling the functions described in the algorithms described in the previous subsections, thus providing the entire pipeline of the recommendation system.

For the main algorithm, the following parameters are required:

- File containing the target subsystem pcsFile;
Algorithm 5: Recommendation pipeline for input requirements - recommendationPipeline().

```
input : Input requirements inputReqs;
   Target specifications specs;
   Type of IoC variation IoC;
   K threshold for top-K recommendations;
   Type of format for recommendations format.
output: Recommendations recommendations.
1 iocDict ← dict();
2 foreach ir ∈ inputReqs do
3   foreach s ∈ specs do
4     // User picks the IoC variation.
5     if IoC = "main" then
6         intersection ← defineIntersection(ir.mainChunks, s.mainChunks);
7         iocVal ← calculateIoC(ir.mainChunks, s.mainChunks, intersection);
8     else
9         intersection ← defineIntersection(ir.chunks, s.chunks);
10        iocVal ← calculateIoC(ir.chunks, s.chunks, intersection);
11     iocDict[s.identifier] ← iocVal;
12 end
13 iocDict ← reduceTopK(iocDict, K) // Reduce to K highest similarity values.
14 recommendations ← buildRecommendations(iocDict, format, recommendations);
15 return recommendations;
```

- K threshold for top-K recommendations;
- The input requirements inputReqs target of recommendations;
- IoC variation (main\text{\_IoC} or chunk\text{\_IoC});
- Format of recommendation outputs (.xlsx or .json) format;
- Minimum support threshold for frequent pattern mining supThreshold;
- Minimum confidence threshold for WARM confThreshold;
- Main chunking threshold ratio for specifications specChunking;
- Main chunking threshold ratio for requirements reqChunking;
- Path of folder folder;

First, we define a Dataset object from the subsystem contained in the file pcsFile (line 1). From this object, we extract the DSs and the CRs of the subsystem, using the functions extractSpecifications and extractRequirements, respectively (lines 2–3). In addition, the grammar and the list stopwords are loaded (lines 4–5). Afterwards, we perform the Preprocessing (Algorithm 1), Main Chunking (Algorithms 2, 3) and Rule and Knowledge Representation (Algorithm 4) tasks independently for DSs (lines 6–8), CRs
Chapter 4: Implementation (lines 9–11), and input requirements (except the last task, lines 12–13). Finally, we provide top-K the recommendations for the input requirements (Algorithm 5) to the engineer (lines 14–15).

**Algorithm 5:** Main algorithm.

```
input : File containing the target subsystem pcsFile;
Input requirement(s) inputReqs;
K threshold for top-K recommendations;
IoC variation (mainIoC or chunkIoC) ioc;
Format of recommendation outputs (xlsx or json) format;
Minimum support threshold for frequent pattern mining supThreshold;
Minimum confidence threshold for weighted association rule mining confThreshold;
Main chunking threshold for specifications specChunking;
Main chunking threshold for requirements reqChunking;
Path of folder folder.
output: Recommendations recommendations.
1 df ← Dataset(pcsFile);
2 specs ← df.extractSpecifications();
3 reqs ← df.extractRequirements(specs);
4 cp ← preprocessing.defineGrammar();
5 stopwords ← preprocessing.defineStopwords();
// Pipeline for DSs.
6 specChunks, specFreqs ← chunkingPipeline(specs, cp, stopwords);
7 specMainChunks ← mainChunkPipeline(specFreqs, specChunking);
8 specMainChunks ← representationPipeline(specMainChunks, specChunks, specs, confThreshold, folder);
// Pipeline for CRs.
9 reqChunks, reqFreqs ← chunkingPipeline(reqs, cp, stopwords);
10 reqMainChunks ← mainChunkPipeline(reqFreqs, reqChunking);
11 reqMainChunks ← representationPipeline(reqMainChunks, reqChunks, reqs, confThreshold, folder);
// Pipeline for input requirements.
12 irChunks, irFreqs ← chunkingPipeline(inputReqs, cp, stopwords);
13 irMainChunks ← mainChunkPipeline(irFreqs, reqChunking);
// Recommendations.
14 recommendations ← recommendationPipeline(inputReqs, specs, ioc, K, format);
15 return recommendations;
```
Chapter 5

Evaluation

In this chapter, we present the iterative experiments and evaluation performed over the raw four PCSs we work with to answer the following subquestions (SQ), linked directly to the Research Questions (Table 5.1) described in Section 3.1.2:

• **SQ1.** Is ARRINA able to process correctly specifications and requirements written in NL?

• **SQ2.** Can the model identify what information has value to stakeholders?

• **SQ3.** Does the model have the capability to define meaningful relations between components of a subsystem?

• **SQ4.** Is it possible to evaluate recommendation performance through the data that already exists?

• **SQ5.** Is ARRINA able to accurately recommend specifications through textual features?

• **SQ6.** Can the tool reduce the effort in linking for engineers?

<table>
<thead>
<tr>
<th>Subquestion</th>
<th>Research Question</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQ1</td>
<td>RQ1</td>
</tr>
<tr>
<td>SQ2</td>
<td>RQ2</td>
</tr>
<tr>
<td>SQ3</td>
<td>RQ3, RQ4</td>
</tr>
<tr>
<td>SQ4</td>
<td>RQ5</td>
</tr>
<tr>
<td>SQ5</td>
<td>RQ5</td>
</tr>
<tr>
<td>SQ6</td>
<td>RQ5</td>
</tr>
</tbody>
</table>

The following sections allow answering these subquestions. For each one, we first describe the configuration and experimentation of the main phases of the model that we tested, then present the results obtained.
### 5.1 Extracting and Processing Requirements

This section describes the evaluation of the capability of the preprocessing phase in interpreting natural language and extracting the DSs and CRs of the subsystems.

In preprocessing, the stopword removal task involved the English stopword corpus defined by NLTK [13]. During the initial experiments, we realized that we needed to remove the word not from the stopwords list since it had an impact on the integrity of the chunks extracted.

Unfortunately, the number of project requirements did not give a complete sense of the size of the subsystems we work with because textual descriptions may vary between them. As a result, many descriptions can be considered negligible, with little information, and vice-versa. So, we also were focused on understanding how many sentences were extracted from the specifications and requirements and their average size.

<table>
<thead>
<tr>
<th>PCS</th>
<th>DS</th>
<th>CR</th>
<th>DS</th>
<th>CR</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>526</td>
<td>164</td>
<td>169</td>
<td>164</td>
</tr>
<tr>
<td>P2</td>
<td>192</td>
<td>284</td>
<td>188</td>
<td>284</td>
</tr>
<tr>
<td>P3</td>
<td>227</td>
<td>288</td>
<td>227</td>
<td>288</td>
</tr>
<tr>
<td>P4</td>
<td>165</td>
<td>169</td>
<td>164</td>
<td>164</td>
</tr>
</tbody>
</table>

Q: Quantity  T: Number of distinct tokens  S: Number of sentences  S\textsubscript{avg}: Average length per sentence

From what is described in Table 5.2, it is possible to infer that the number of sentences fluctuates through the various PCSs, while the average sentence length is similar within each domain, considering the number of characters. However, it is possible to observe the impact of the preprocessing methods on each variable in the analysis. The number of tokens, the number of sentences, and the average length of sentences are reduced due to the removal of non-specifications and stopwords in each sentence (columns 3–5, 7–9, 11–13, 15–17). Moreover, derived from this, the number of specifications is also significantly reduced in P1 and P4 (before preprocessing, column 2; after preprocessing, column 10). This reduction proves that the model can process DSs and CRs, identify textual descriptions that do not correspond to specifications, and correctly process them, answering thus positively, to SQ1.

### 5.2 Main Chunk Identification and Relevancy

In this section, we present the experimentation and evaluation performed over the main chunking task to identify relevant components in DSs and CRs of the PCSs. In addition, we experimented with the threshold that defines main chunks and performed alterations in the grammar given the relevancy of the main chunks obtained. These experimentations were performed and validated on the P1 subsystem and
then propagated to other PCSs ($P_2, P_3, P_4$).

### 5.2.1 Optimal Threshold

Noun Phrase Chunking step uses the POS tagged sentences and the grammar described in Table 3.3 as input. For each sentence, the grammar parser is run and extracts all recognizable noun phrase chunks (*chunks* for short). For each chunk is also obtained its frequency.

To understand which is the optimal threshold to extract main chunks, we tried thresholds between 3 to 6, and we used the $P_1$ subsystem as reference (Table 5.3, columns 2–3, for $P_1$). Table 5.4, columns 2–5, 10–13, state the number of chunks and data about their frequency. The number of chunks ($|ch|$) is relatively high in every PCS, with many chunks with minimal frequency, and so its average in the whole chunk set is low.

After some analysis and discussion with experts over $P_1$, the optimal threshold was defined as 4. Therefore, chunks with a frequency greater than or equal to this threshold were considered as main chunks. This threshold accounts for chunks that occur in at least 2.36% of the DSs and 2.43% of the CRs of this PCS (third line of the Table 5.3), and this is the ratio we used for the other subsystems. Table 5.4, columns 6–9, 14–17, provide the difference in terms of the number of main chunks for the thresholds introduced earlier in this section: higher thresholds than 4 provide small sets of main chunks in $P_3$ and $P_4$, making analysis more difficult. In comparison, smaller thresholds provide chunks irrelevant to engineers.

### 5.2.2 Main Chunk Relevancy

We decided to use PCS human expert validation to evaluate each grammar rule based on the relevancy of each extracted main chunk. To do so, for each main chunk, we asked if it was considered *highly relevant* (HR), i.e., recognized as a PCS component, *possibly relevant* (PR), i.e., considered as a state, or *not relevant* (NR).

This process was done in two iterations for $P_1$, and the resulting relevancy distribution is described in Table 5.5, columns 4–6. In the first iteration (line 2), with an initial grammar containing the first three rules of Table 3.3 and only analyzing specifications, 93, 9% (62) of the 66 extracted main chunks from DS were considered *highly or possibly relevant* by PCS experts. After discussion with them, we added a new grammar rule (last line of Table 5.5) to capture new, interesting patterns. As stated in line 3, from the 100 main chunks extracted in this iteration, 94% (89) were considered *highly or possibly relevant* by PCS experts. Although there is a reduction in the ratio of *highly relevant* main chunks when adding the new rule, we decided that the grammar is helpful as it was able to extract 25 new *highly relevant* main chunks that are helpful to decision-makers.

Afterwards, we applied the same evaluation process over CRs. We observed that the results were similar to the ones obtained over the specifications, in the second iteration, since 92, 2% of the main
Table 5.3: Comparison of patterns and rules extracted for different main chunking thresholds.

<table>
<thead>
<tr>
<th>PCS</th>
<th>T</th>
<th>%DS</th>
<th>%CR</th>
<th>DS</th>
<th>CR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1</td>
<td>3</td>
<td>1.77</td>
<td>1.83</td>
<td>137</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>2.36</td>
<td>2.43</td>
<td>214</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>2.95</td>
<td>3.05</td>
<td>184</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>3.55</td>
<td>3.66</td>
<td>151</td>
<td>33</td>
</tr>
<tr>
<td>P2</td>
<td>3</td>
<td>1.77</td>
<td>1.83</td>
<td>175</td>
<td>73</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>2.36</td>
<td>2.43</td>
<td>182</td>
<td>86</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>2.95</td>
<td>3.05</td>
<td>223</td>
<td>205</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>3.55</td>
<td>3.66</td>
<td>204</td>
<td>187</td>
</tr>
<tr>
<td>P3</td>
<td>3</td>
<td>1.77</td>
<td>1.83</td>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>2.36</td>
<td>2.43</td>
<td>17</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>2.95</td>
<td>3.05</td>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>3.55</td>
<td>3.66</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>P4</td>
<td>3</td>
<td>1.77</td>
<td>1.83</td>
<td>22</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>2.36</td>
<td>2.43</td>
<td>22</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>2.95</td>
<td>3.05</td>
<td>14</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>3.55</td>
<td>3.66</td>
<td>14</td>
<td>0</td>
</tr>
</tbody>
</table>

T: P1 Threshold
P: Patterns
%: Percentage of requirements
R: Rules

Table 5.4: Chunks, frequency, and main chunking extraction for various thresholds.

<table>
<thead>
<tr>
<th>PCS</th>
<th>DS</th>
<th>CR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1</td>
<td>1786</td>
<td>910</td>
</tr>
<tr>
<td>P2</td>
<td>2342</td>
<td>1604</td>
</tr>
<tr>
<td>P3</td>
<td>860</td>
<td>252</td>
</tr>
<tr>
<td>P4</td>
<td>256</td>
<td>645</td>
</tr>
</tbody>
</table>

[ch]: Number of chunks
f_{int}: Frequency interval
f_{avg}: Frequency average
f_{dev}: Frequency standard deviation
T_{int}: P1 main chunking threshold
Table 5.5: Main chunk proportions extracted with each grammar, for $P_1$, in each validation iteration.

<table>
<thead>
<tr>
<th>I</th>
<th>Grammar</th>
<th>DS</th>
<th>CR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>[ch]</td>
<td>HR (%)</td>
</tr>
<tr>
<td>1</td>
<td>${&lt;JJ&gt;+&lt;NNP&gt;+}&lt;NN&gt;<em>&lt;NNP&gt;+&lt;NN&gt;</em>}&lt;JJ&gt;+&lt;NN&gt;} $</td>
<td>66</td>
<td>78.8</td>
</tr>
<tr>
<td>2</td>
<td>${&lt;JJ&gt;+&lt;NNP&gt;+}&lt;NN&gt;<em>&lt;NNP&gt;+&lt;NN&gt;</em>}&lt;JJ&gt;+&lt;NN&gt;} $</td>
<td>100</td>
<td>77</td>
</tr>
</tbody>
</table>

$I$: Iteration  
[ch]: Number of extracted main chunks  
HR(%) : % of highly relevant main chunks extracted  
PR(%) : % of possibly relevant main chunks extracted  
NR(%) : % of not relevant main chunks extracted

Chunks were considered highly or possibly relevant. We also observed that 38% of highly relevant main chunks extracted in DSs occur in CRs.

Evaluating the relevancy in the other subsystems (Table 5.6) using the same set of grammar rules of the second iteration, we verified that the percentage of HR and PR main chunks is similar to the ones obtained in $P_1$ for specifications. However, there is a slight degradation of performance for requirements, which can be explained by the small number of requirements and the low number of chunks, leading to a higher percentage of irrelevant main chunks obtained.

Table 5.6: Chunk relevancy distribution for subsystems $P_2$, $P_3$ and $P_4$.

<table>
<thead>
<tr>
<th>PCS</th>
<th>DS</th>
<th>CR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[ch]</td>
<td>HR (%)</td>
</tr>
<tr>
<td>$P_2$</td>
<td>82</td>
<td>83.0</td>
</tr>
<tr>
<td>$P_3$</td>
<td>23</td>
<td>82.6</td>
</tr>
<tr>
<td>$P_4$</td>
<td>45</td>
<td>73.3</td>
</tr>
</tbody>
</table>

[ch]: Number of extracted main chunks  
HR(%) : % of highly relevant main chunks extracted  
PR(%) : % of possibly relevant main chunks extracted  
NR(%) : % of not relevant main chunks extracted

Given the experiments provided in this section, it is possible to answer to SQ2. correctly: the model can identify, through an integer threshold, valuable chunks to an engineer, given their high frequency along DSs or CRs. Based on the expert’s evaluation, it was also possible to notice that a high percentage of the main chunks obtained in each PCS was considered relevant to the testing process. As a consequence, frequency-based refinement is enough to filter components of interest.

5.3 Knowledge Subsystem Representation

In order to represent the knowledge of the subsystems, regarding their components (represented by main chunks), it contains and their associations, we compared WARM with ARM derived from the application of the Rule and Knowledge Representation task. Again, we took as reference the $P_1$ subsystem.
For ARM, we extracted frequent itemsets without using the *weight* parameter to understand which patterns would be extracted without the impact of the derived chunks. Frequent itemsets from DSs were extracted with a minimum support threshold of 5 transactions (≈ 2.9%), and then rules were extracted for a minimum confidence threshold of 50% (Table 5.7, lines 2–6, 12–16). We manage to extract 428 itemsets (columns 3–5, lines 2–6) and 459 rules (columns 6–9, lines 2–6) from DSs, many with the highest possible confidence ($C = 1$), which did not allow to get conclusions about main chunk relations. Next, to better understand the model behaviour, we retrieved the rules with WARM (Table 5.7, lines 7–11, 17–21), in which we reduced the support threshold to 0.5% to capture a significant set of rules, and kept the same minimum confidence threshold. WARM retrieved 214 *weighted* itemsets (columns 3–5, lines 7–11) and 50 *weighted* rules (columns 6–9, lines 7–11). These resulting rules are better related to the context of the problem because weight depends on derived chunks and their uniqueness value (UV). Therefore, we perform the same comparison in the other PCSs, with results described in Table A.1, Table A.2 and Table A.3 of Appendix A. We disclose some of the names of the main chunks we represent in association rules and knowledge graphs throughout this document for intellectual property reasons.

Table 5.7: Itemsets and rules extracted in P1 with ARM and WARM.

<table>
<thead>
<tr>
<th>D</th>
<th>F</th>
<th>Items</th>
<th>S_i</th>
<th>#T</th>
<th>Association Rules</th>
<th>S_r</th>
<th>C</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>DS</td>
<td>ARM</td>
<td>(MC)</td>
<td>0.221</td>
<td>57.00</td>
<td>(MC) ⇒ (MC2)</td>
<td>0.070</td>
<td>1.00</td>
<td>9.92</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(AC)</td>
<td>0.186</td>
<td>48.00</td>
<td>(MC, MC/AUX) ⇒ (AUX)</td>
<td>0.050</td>
<td>1.00</td>
<td>5.38</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(effort reference)</td>
<td>0.019</td>
<td>5.00</td>
<td>(DC, MC/AUX) ⇒ (MC)</td>
<td>0.020</td>
<td>0.50</td>
<td>2.26</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(TCMS, MC, Converter, MVB)</td>
<td>0.019</td>
<td>5.00</td>
<td>(TCMS, MC) ⇒ (MC)</td>
<td>0.020</td>
<td>0.50</td>
<td>2.26</td>
</tr>
<tr>
<td></td>
<td>WARM</td>
<td>(AUX)</td>
<td>0.254</td>
<td>42.96</td>
<td>(MC, MC/AUX) ⇒ (AUX)</td>
<td>0.008</td>
<td>0.99</td>
<td>3.95</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(MC)</td>
<td>0.213</td>
<td>36.09</td>
<td>(MC/AUX) ⇒ (AUX)</td>
<td>0.016</td>
<td>0.95</td>
<td>3.65</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(AC, OVP)</td>
<td>0.005</td>
<td>0.85</td>
<td>(DC, AUX) ⇒ (MC)</td>
<td>0.020</td>
<td>0.52</td>
<td>2.39</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(AUX, DC-link, MC)</td>
<td>0.005</td>
<td>0.85</td>
<td>(AUX, MVB, Converter) ⇒ (MC)</td>
<td>0.006</td>
<td>0.51</td>
<td>2.39</td>
</tr>
<tr>
<td>CR</td>
<td>ARM</td>
<td>(propulsion system)</td>
<td>0.237</td>
<td>39.00</td>
<td>(RTS) ⇒ (LC)</td>
<td>0.054</td>
<td>1.00</td>
<td>14.91</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(LC)</td>
<td>0.219</td>
<td>36.00</td>
<td>(LC, line voltage) ⇒ (current limit)</td>
<td>0.054</td>
<td>1.00</td>
<td>10.25</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(unit, MVB)</td>
<td>0.005</td>
<td>0.03</td>
<td>(Converter, LC) ⇒ (AC)</td>
<td>0.030</td>
<td>0.50</td>
<td>2.10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(main transformer, LC)</td>
<td>0.005</td>
<td>0.03</td>
<td>(Converter, LC) ⇒ (Motor Converter)</td>
<td>0.030</td>
<td>0.50</td>
<td>2.10</td>
</tr>
<tr>
<td></td>
<td>WARM</td>
<td>(AC)</td>
<td>0.152</td>
<td>24.98</td>
<td>(Converter, MC) ⇒ (AC)</td>
<td>0.005</td>
<td>0.79</td>
<td>3.25</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Converter)</td>
<td>0.149</td>
<td>24.49</td>
<td>(Converter, MC) ⇒ (AC)</td>
<td>0.019</td>
<td>0.78</td>
<td>5.10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(DMC)</td>
<td>0.005</td>
<td>0.82</td>
<td>(MC, LC) ⇒ (AC)</td>
<td>0.009</td>
<td>0.55</td>
<td>3.58</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(AC, Converter, damp link)</td>
<td>0.005</td>
<td>0.82</td>
<td>(MC) ⇒ (AC)</td>
<td>0.040</td>
<td>0.54</td>
<td>3.57</td>
</tr>
</tbody>
</table>

D: Domain F: Framework S_i: Itemset Support #T: Number of Transactions S_r: Rule Support C: Confidence L: Lift

Figure 5.1 shows the knowledge graphs with the association rules between components extracted from DSs and CRs in P1, for a minimum $WC$ threshold of 10.0%: if $WC > 80\%$, a red-colored edge is returned; if $50\% \leq WC < 80\%$, an orange-red-colored edge is returned; if $20\% \leq WC < 50\%$, an orange-colored edge is returned; and if $WC \leq 20\%$, a pink-colored edge is returned. Looking at both graphs, we can affirm that the main chunks are reduced to a cohesive nucleus of components containing relations between them. It is also possible to see strong connections between components with a high number of derived chunks. Knowledge graph for the other PCSs we work with are depicted in Figure A.1, Figure A.2, and Figure A.3 of Appendix A.
(a) Components in design specifications. (b) Components in customer requirements.

Figure 5.1: Knowledge graphs for DSs and CRs of subsystem P1.

To verify the experimentation related to rule mining, we propagated the rule framework to the other PCSs, with results depicted in Figure 5.2. Subsystems P1 and P2 can extract many patterns and rules, while the opposite occurs in subsystems P3 and P4. These two affirmations lead us to believe that the first two subsystems have strong connections between components. At the same time, the last two do not. From the rules and patterns extracted, engineers can infer if components inside the DSs and CRs are linked as expected and whether the information provided in textual descriptions is sufficient for testing quality.

Figure 5.2: Frequent patterns and association rules extracted for each PCS.

We also present the results for different main chunking thresholds in Table 5.3, aiming to analyze tendencies between the patterns and rules extracted for different thresholds. There are no clear tendencies regarding the increase or decrease of patterns and rules between different thresholds. Nonetheless,
changing thresholds does not improve the extraction in $P_3$ and $P_4$, solidifying our assumption that the domains have few relations between their components.

Through the analysis described above, we can affirm that main chunks better represent relevant components and, therefore, the knowledge that should be considered essential to the testing process. In addition, such results allow us to answer positively to SQ3. Furthermore, we believe that the main chunking and the WARM frameworks allow for refining association rules, reducing noise obtained from earlier tasks in the pipeline.

### 5.4 Inter- and Intra-domain Recommendations

Following the reasoning described in RQ5 of Section 3.1.2, we evaluated recommendations for input requirements in two different frameworks: (1) the inter-domain framework (Section 5.4.1), where we evaluated specification recommendations based on already established connections inside the various PCS; and (2) the intra-domain framework (Section 5.4.2), where we recommend main transformer requirements that were never connected to any PCS (engineers are responsible for validation of the results).

#### 5.4.1 Inter-domain Framework

To understand if the top-K recommendations provided to input requirements were accurate and precise, it was required to evaluate them over the already existing PCSs and their specification-requirement connections. Right away, we state a set of concepts associated with our evaluation framework, which will be used frequently through this section:

- **Accurate link**: a link that falls into the top-K recommendations.
- **Positive link**: a link with a positive $main_{ioc}$ or $chunk_{ioc}$.
- **Negative link**: a link with $main_{ioc}$ and $chunk_{ioc}$ both null.
- **Non-Textual link**: a link where one element (specification or requirement) refers to a Table, Figure, Equation, or an external document.

Our evaluation focused on the following topics:

- To observe accuracy and precision variations in pipelines with and without non-specification identification;
- To observe the accuracy and precision variation between pipelines with the similarity between chunks and similarity between main chunks;
• To identify how many specification-requirement links have chunks identified;
• To understand which specification-requirement links have references to other documents or visual elements;
• To visualize the accuracy and precision variation between different values of K;
• To tune the main chunking thresholds and understand the variation of accuracy for mainioc.

Next, we discuss the assessments made over these six points.

**Accuracy and precision in non-specification identification pipelines.**

Analyzing the pipelines with and without non-specification identification, we evaluated recommendations provided with $K = 5$ and chunkioc, in three settings: S1, non-specifications are not identified and, therefore, are preprocessed; S2, we only removed specifications that referred to comments; S3, we perform full non-specification identification, and remove textual specifications identified in the first step of Section 3.3.1. In every setting, we extracted the number of accurate links to observe its variation. Figure 5.3 depicts the results obtained: in S3, the accuracy results are higher in the P1 and P4 subsystems. This occurs because, as stated in Table 5.2, non-specification identification recognizes many contents that are not noted as DSs, which reduces the candidates to be recommended. Therefore, we consider non-specification identification can help achieve better accuracy.

![Figure 5.3: Accurate recommendations in each non-specification identification, $K = 5$.](image)

**Accuracy and precision variation between pipelines with chunkioc and with mainioc.**

ARRINA can perform recommendations resorting to chunks (chunkioc metric) or main chunks (mainioc metric). At first, we believed that only using chunkioc would lead to interesting results since a link can
be identified by the keywords each element contains. However, given that \( \text{main}_{\text{ioc}} \) is essential to extract relevant findings of relations inside domains, we wanted to observe accuracy results through this metric.

As described in Figure 5.4, for \( K=5 \), \( \text{main}_{\text{ioc}} \) poorly approximates to the results obtained with \( \text{chunk}_{\text{ioc}} \) in every PCS. This certifies the potentialities of token-based similarity methods since \( \text{chunk}_{\text{ioc}} \) presents close to 90% in some subsystems, and yet, also certifies that the main chunking framework can lead to a degree of loss of information, which leads to poorer results. This observation gives us the sense that the \( \text{main}_{\text{ioc}} \) metric can be more useful for analysis inside the domain instead of analysis between PCSs.

![Comparison of accurate recommendations with mainioc and chunkioc, K = 5.](image)

**Figure 5.4: Comparison of accurate recommendations with mainioc and chunkioc, \( K = 5 \).**

Identification of chunks in specification-requirement links.

Based on cases defined in Table 3.11, we divided the links in two types: positives and negatives. We establish positive links as the ones that the recommendation system can identify, given there is some level of similarity between its elements (i.e., one of the metrics has a positive value). Negative links mean the opposite. The system needs to identify as many positive links as possible. From the visualization described in Figure 5.5, it is perceived that the amount of links considered positive is considerably higher than the number of negative ones (except in \( P3 \)), with percentages over 80%. Observing these percentages, we can conclude that the system can identify similarities between specifications and requirements correctly already linked.

Identification of references in connections to non-textual elements.

After discussion with engineers, it was defined that the system should not evaluate connections in which one of the elements references a non-textual element, such as external documents, tables, figures or equations. These elements might provide value to the analysis that our system cannot perform. Figure 5.6
shows the ratio of non-textual links in the whole set, for each PCS. Even though this ratio is small, this analysis is relevant since the reference to external elements leads to readability loss, which also affects the performance of ARRINA.

**Accuracy variation in different variations of K.**

To deepen the analysis of the \( \text{chunk}_{\text{ioc}} \) and \( \text{main}_{\text{ioc}} \) metrics and to understand which values of K could be understood as optimal, and we extracted the recommendation results in \( K \in [1, 15] \). We analyzed the results in terms of variation of K per metric, with the following observations: as previously noted,
chunkioc performs better than mainioc overall, due to the degree of information lost in the main chunking process. As also expected, as K grows (Table 5.8, Figure 5.7), the accuracy gets higher as well. This phenomenon is especially relevant with the chunkioc metric: in every PCS, with $K > 8$, the accuracy obtained is higher than 50%, with perfect results in P3 as well.

Figure 5.8 depicts a comparison, associated with trendlines, of accuracy between the two IoC variations - chunkioc and mainioc - for each PCS. Every variation of K is followed by a logarithmic trendline, which shows that whether K is high, the growth rate slows down as it is tougher to retrieve correct candidates. Also, it is observed once again that mainioc fails to approximate to chunkioc, which allows us to reinforce that the former can be more useful for analysis within the domain.

### Table 5.8: Accuracy per K threshold.

<table>
<thead>
<tr>
<th>Metric</th>
<th>PCS</th>
<th>Accuracy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>P1</td>
<td>4.44 8.33 13.89 18.33 19.44 23.89 26.11 31.11 33.33 34.44 36.67 40.00 40.56 47.22 48.33</td>
</tr>
<tr>
<td></td>
<td>P2</td>
<td>5.75 9.32 13.97 18.08 22.19 26.85 30.14 32.88 35.89 37.26 40.00 41.10 43.01 44.38 45.48</td>
</tr>
<tr>
<td></td>
<td>P3</td>
<td>9.30 16.28 23.26 30.23 37.21 44.19 48.84 53.49 55.81 60.47 65.12 67.44 67.44 69.77 74.42</td>
</tr>
<tr>
<td></td>
<td>P4</td>
<td>31.75 41.27 51.59 55.56 58.73 59.52 61.90 65.08 66.67 67.46 69.05 71.43 72.22 72.22 72.22</td>
</tr>
<tr>
<td>mainioc</td>
<td>P1</td>
<td>10.00 18.33 25.00 32.78 37.78 41.11 45.00 50.00 53.33 55.00 56.11 57.78 60.56 61.67 63.33</td>
</tr>
<tr>
<td></td>
<td>P2</td>
<td>18.36 24.38 31.51 38.90 42.19 46.03 47.95 52.33 54.25 57.26 58.90 61.10 63.29 63.84 64.38</td>
</tr>
<tr>
<td></td>
<td>P3</td>
<td>44.19 62.79 74.42 83.72 90.70 93.55 97.67 100 100 100 100 100 100 100 100 100</td>
</tr>
<tr>
<td></td>
<td>P4</td>
<td>59.52 73.02 76.19 79.37 84.13 84.13 86.51 87.30 88.10 88.89 90.48 91.27 91.27 91.27 93.65</td>
</tr>
<tr>
<td>chunkioc</td>
<td>P1</td>
<td>10.00 18.33 25.00 32.78 37.78 41.11 45.00 50.00 53.33 55.00 56.11 57.78 60.56 61.67 63.33</td>
</tr>
<tr>
<td></td>
<td>P2</td>
<td>18.36 24.38 31.51 38.90 42.19 46.03 47.95 52.33 54.25 57.26 58.90 61.10 63.29 63.84 64.38</td>
</tr>
<tr>
<td></td>
<td>P3</td>
<td>44.19 62.79 74.42 83.72 90.70 93.55 97.67 100 100 100 100 100 100 100 100 100</td>
</tr>
<tr>
<td></td>
<td>P4</td>
<td>59.52 73.02 76.19 79.37 84.13 84.13 86.51 87.30 88.10 88.89 90.48 91.27 91.27 91.27 93.65</td>
</tr>
</tbody>
</table>

### Tuning the main chunking framework.

To understand the degree of separation between the metrics mainioc and chunkioc, we tuned the main...
Figure 5.8: Accuracy comparison of IoC metrics for each PCS, $K \in [1, 15]$. 
Table 5.9: Comparison of main_{ioc} accuracy for different main chunking thresholds, $K = 5$.

chunking threshold from the optimal value determined in Section 5.2.1. As a reference, the main chunking threshold for $P_1$ is 4, accounting for 2.36% of DSs and 2.43% of CRs on that PCS. We decided to tune this threshold to the values perceived in Table 5.9 and Figure 5.9. It should be noted that we do not evaluate the relevancy of the new main chunks extracted for lower thresholds.

From Table 5.9 and Figure 5.9, it is possible to verify that main_{ioc} has a tendency to decrease as the main chunking threshold increases. This is likely because there are fewer main chunks extracted, leading to a higher degree of loss of information, and hence reducing the metric accuracy. Comparing a threshold of $(\%_{DS}, \%_{CR}) = (2.36, 2.43)$ with a threshold of $(\%_{DS}, \%_{CR}) = (3.55, 3.66)$, accuracy decrease is of 37.9% in $P_4$ (lines 9, 17), and 13.95% in $P_3$ (lines 8, 16). The accuracy decrease is not linear, however, in subsystem $P_2$, for example, the accuracy increases from $(\%_{DS}, \%_{CR}) = (2.36, 2.43)$ to $(\%_{DS}, \%_{CR}) = (2.95, 3.05)$ (lines 7, 11), which gives the sense that even with the main chunk reduction associated with a threshold increase, removing noise can lead to better performance.

Based on the experiments we did on the six topics mentioned above and the conclusions we drew, we answer to SQ5. positively and with confidence in our proposed recommendation system.

5.4.2 Intra-domain Framework

To understand the capability of the recommendation system in evaluating input requirements, we used a set of Main Transformer requirements as input requirements. From these requirements, the system
outputted top-5 recommendations, which an engineer evaluated.

As depicted in Figure 5.10, for a set of main transformer requirements and a set of DSs of a PCS, the engineer receives the DS recommendations given by the ARRINA recommendation system for each input requirement. The engineer is responsible for the analysis of the links in each recommendation, splitting each into four possible categories, as described in Table 5.10. This evaluation was performed for subsystems P2 and P3 since these are directly connected to the Main Transformer.

Table 5.10: Evaluation categories for a specification-requirement link.

<table>
<thead>
<tr>
<th>Category</th>
<th>Correlation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-10%</td>
<td>Bad</td>
<td>System fails to recommend an interesting link.</td>
</tr>
<tr>
<td>10-50%</td>
<td>Mediocre</td>
<td>System recognizes some similarity.</td>
</tr>
<tr>
<td>50-85%</td>
<td>Good</td>
<td>System fails to get the most accurate link, but it is close.</td>
</tr>
<tr>
<td>85-100%</td>
<td>Very Good</td>
<td>System recognizes the most accurate link.</td>
</tr>
</tbody>
</table>

We break down evaluation as follows: we classify the link with the highest correlation for each top-K recommendation. For example, for an input requirement, if two links have bad correlation, two have mediocre correlation and one has good correlation, we mark this recommendation as good, since this was the highest correlation found. This process emulates well how an engineer can pick a link among K in a top-K recommendation since this person will choose the DSs that will correlate the most to the input requirement. Lastly, we gathered the number of links was represented in each category.

Through the bar plots illustrated in Figure 5.11, it is possible to observe the distribution of the number of links in each category of each PCS evaluated by the engineer. In P2, 93.5% of the recommendations are considered good or very good, while in P3, 69.5% fall into the same categories. We believe these are relevant results that state the efficiency of the recommendations to never before linked requirements.

However, there is still a proportion of links, especially in P3, that fall into the bad and mediocre
categories. After some analysis and discussion with the engineer that evaluated recommendations, we point out why the system fails in providing relevant links:

- **The system has difficulty in capturing components that are more generic or are constituted by a common noun.** Since we rely upon CFGs to extract components, this method can present limitations. For example, the grammar we use cannot extract components such as "oil" or "fan", only if those are connected to another noun or a proper noun.

- **Lack of writing standards.** Dealing with NL with no restrictions might cause incongruities in writing for specifications and specifications. From example, DC-link ≠ DC 1ink, or backup ≠ back-up ≠ back up. Cases as these occur frequently along the subsystems we work with and degrade the calculation of similarity between chunks. This issue might be solved with the introduction of lemmatization techniques in the preprocessing phase of ARRINA.

- **Some components can be referred through various synonyms.** Many components identified are
Chapter 5 Evaluation

referred with a particular term in a specification, while in a requirement are mentioned with another. Since the system does not use, a priori, any form of domain knowledge to infer relationships, token-based methods may lose performance.

However, we can assert that through the findings in Section 5.4.1 and Section 5.4.2 it is possible to answer to questions SQ4. and SQ5. The inter-domain recommendation framework proved that the recommendations obtained were close to the performance of engineers, providing a reduction of costs in linking and testing processes with good accuracy results in some subsystems. From the intra-domain recommendations, we can also affirm that the system can link requirements that were never associated with any specification. The engineers defined many of those recommendations as good or very good.

5.5 Key Performance Indicator Analysis

This section is focused on understanding if system performance successfully achieves the objectives proposed in the Key Performance Indicators (KPI) described throughout the XIVT project for Bombardier Transportation. For that, we evaluated accuracy and time of analysis and checked if these fulfilled the fixed goals. Also, in this section, we aim to provide a complete analysis of the human methods and the methods provided by our system to recommend possible links. This evaluation is focused on the following topics:

- Formulation of the time it takes for engineers and for the tool to analyze requirements;
- Comparison of times with the PCSs used in this work;
- Comparison of time growth rate for different numbers of input requirements;
- Analysis of the impact of accuracy in time of analysis.

Formulation of times of analysis.

Reduction of human effort is essential to define the performance of the system. As determined by the engineers, in the worst-case scenario, on average, 20 seconds is the time required to analyze a link, i.e., the time the engineer takes to analyze a possible association between a specification \( sp \in S \) and an input requirement \( ir \in IReq \). Therefore, we define AnalysisTime for a set of specifications \( S \) and input requirements \( IReq \) as the product of their sizes and the 20 seconds of analysis for a pair.

\[
\text{AnalysisTime}(|S|, |IReq|) = |S| \times |IReq| \times 20 \quad (s)
\]  

Equation 5.2 illustrates an example with \( |S| = |IReq| = 200 \), where AnalysisTime = 800 000 seconds, which corresponds to 222.22 hours, or close to 27.8 days of work (assuming a 8-hour workday).
AnalysisTime(200, 200) = 200 \times 200 \times 20 = 800,000 \quad (5.2)

ToolAnalysisTime (Equation 5.3), or the time that an engineer takes to evaluate links in a PCS with the ARRINA system, is more complex to assess, since it depends on the following parameters:

1. The accuracy of the system, ToolAcc, i.e., the ratio of accurate recommendations it can produce;
2. The number of input requirements, |IReq|;
3. K, the number of recommendations to be outputted for an input requirement;
4. The number of specifications, |S_c|. This number excludes the textual descriptions removed on the non-specification identification task of the system.

\[
\text{ToolAnalysisTime}(|IReq|, K, S_c, \text{ToolAcc}) = (1 - \text{ToolAcc}) \times \text{AnalysisTime}(S_c, |IReq|) \\
+ \text{ToolAcc} \times \text{AnalysisTime}(K, |IReq|) \quad (s) \quad (5.3)
\]

This formulation considers two main factors: the time required to analyze expected accurate recommendations (this is the factor most responsible for reducing the time) and the time required to analyze expected inaccurate recommendations manually.

While |IReq| and K do not require any calculation, ToolAcc needs assessment beforehand. To do so, for ARRINA, we specified K = 5 and ToolAcc = 90\% (the best accuracy the system can retrieve along with the subsystems in this work).

It should be also known that the cost required to understand if ARRINA performs correct recommendations on a PCS, CorrectCost (Equation 5.4), depends on the following factors:

- The probability of the tool working with a subsystem, CorrectProb;
- K, the number of recommendations to be outputted for an input requirement;
- ReqEffort, i.e., the number of recommendations required for an engineer to understand if the ARRINA system works well.

\[
\text{CorrectCost}(\text{CorrectProb}, K, \text{ReqEffort}) = \text{CorrectProb} \\
\times \text{AnalysisTime}(K, \text{ReqEffort}) \quad (s) \quad (5.4)
\]

For CorrectProb, we assume that the system produces meaningful recommendations if it provides accuracy above 80\%. As half of the subsystems analyzed in this work (P3, P4) have an accuracy above
80%, therefore, we fixed \( \text{CorrectProb} = 50\% \). Through conversations with engineers, we estimate \( \text{ReqEffort} = 10 \). We calculate \( \text{CorrectCost} \) (Equation 5.5) with these parameters in mind.

\[
\text{CorrectCost}(0.5, 5, 10) = 0.5 \times 5 \times 10 \times 20 = 500
\] (5.5)

Therefore, the total time (\( \text{ToolTime} \)) of the process consisting of execution, analysis and linking is described in Equation 5.7, where in addition to the variables described before, \( \text{DiscardedTime} \) (Equation 5.6) defines the analysis of the textual descriptions discarded (\( S_{\text{discarded}} \)) in non-specification identification (see Section 3.3.1), and \( \text{ExecutionTime} \) corresponds to ARRINA execution time in a PCS.

\[
\text{DiscardedTime} = \text{AnalysisTime}(S_{\text{discarded}}, I_{\text{Req}}) \quad (s) \] (5.6)

\[
\text{ToolTime} = \text{ToolAnalysisTime} + \text{DiscardedTime} + \text{ExecutionTime} + \text{CorrectCost} \quad (s) \] (5.7)

Comparison of process times.

For each PCS, we calculated the \( \text{AnalysisTime} \) and each factor necessary to calculate \( \text{ToolTime} \) for \( K = 5 \), as described in Table 5.11. \( \text{ExecutionTime} \) and \( \text{CorrectCost} \) have a residual impact in the calculation of \( \text{ToolTime} \).

<table>
<thead>
<tr>
<th>PCS</th>
<th>ToolAnalysisTime (h)</th>
<th>DiscardedTime (h)</th>
<th>ExecutionTime (h)</th>
<th>CorrectCost (h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P_1 )</td>
<td>5.47</td>
<td>91.23</td>
<td>0.020</td>
<td>0.136</td>
</tr>
<tr>
<td>( P_2 )</td>
<td>5.95</td>
<td>1.02</td>
<td>0.046</td>
<td></td>
</tr>
<tr>
<td>( P_3 )</td>
<td>6.95</td>
<td>0.0</td>
<td>0.003</td>
<td></td>
</tr>
<tr>
<td>( P_4 )</td>
<td>3.17</td>
<td>21.98</td>
<td>0.002</td>
<td></td>
</tr>
</tbody>
</table>

In Figure 5.12, we compare \( \text{AnalysisTime} \) and \( \text{ToolTime} \) for every PCS and the set of input requirements, and with \( \text{ToolAcc} \) as previously described. There is a considerable reduction in time, as described by the bar plots. From the line plots, it is also verifiable that the variations in \( \text{AnalysisTime} \) depend not only on the original set of specifications but also on the number of refined specifications. Therefore, \( \text{ToolTime} \) is reduced at different rates. In \( P_1 \), for example, the degree of reduction is not as considerable as other subsystems since the number of discarded specifications is high. Therefore, the engineer must take many hours analyzing these. In \( P_3 \) there is a notable reduction between \( \text{AnalysisTime} \) and \( \text{ToolTime} \), since no specifications were discarded, and so, \( \text{DiscardedTime} = 0 \).

Comparison of time growth rates.

It is also essential to compare \( \text{AnalysisTime} \) and \( \text{ToolAnalysisTime} \) while the number of input requirements increases to understand how much both times grow. Figure 5.13 illustrates a comparison
of time growth, in hours, for four different analysis: AnalysisTime, with 50, 100 and 200 DSs, and ToolAnalysisTime, with $K = 5$, also for 50, 100 and 200 DSs. We observed the time variation with the following numbers of input requirements: 1, 10, 100, 500, 1000. From this figure, it can be observed that all times grow linearly. However, when analyzing recommendations with the tool, the growth rate is substantially lower than the increase when using traditional methods. At the same time, an increase in design specifications means that AnalysisTime and ToolAnalysisTime will grow faster.

Figure 5.13: Comparing AnalysisTime and ToolAnalysisTime analysis time for different values configuration of DS and IReq.
**Parameter in time of analysis.**

We believe that the number of specifications, input requirements and the value of K are essential factors that impact the time it takes to analyze recommendations. Though a high value of K can provide an excellent accuracy value, the trade-off between the metric and the time required for analysis of the recommendations provided can be unbearable for the engineer or even approximate to the traditional time to analyze recommendations. Therefore, we decided to tweak $S$, $I Req$ and $K$ to determine how these parameters would change $ToolAnalysisTime$.

Figure 5.14 illustrates the variation of analysis of time with variation of these parameters, from 0 to 100. Right away, it is possible to see that variation in $K$ is more impactful than variation in $S$ and $IR$. This occurs since, as seen in Equation 5.3, in high accuracy, the growth in K has more impact in the calculations than growth in the number of specifications involved in the analysis. As expected, with high values of K, $ToolAnalysisTime$ quickly approximates to $AnalysisTime$.

![Figure 5.14: Parameter variation in ToolAnalysisTime.](image)

From the evaluation performed above, we can positively respond to SQ6. Furthermore, for high levels of accuracy, the recommendation system dramatically reduces the time required for link analysis. While in some subsystems, the engineer would require close to 50 hours of work, through the tool, for $K = 5$, there is an evident reduction to approximately 7 hours of work. Thus, on average, an engineer can expect a reduction of 36 hours of work with the integration of ARRINA. This phenomenon leads us to believe that engineers can have their effort considerably reduced and the KPI objectives reached. Therefore, our tool can be integrated within the quality testing methodologies.
Chapter 6

Conclusion

The work we produced aimed to contribute to research on the use of NLP to automate and reduce the human effort in identifying similarities between project requirements through the ARRINA system. This system recommends software design specifications to input requirements. Therefore, we focused on extracting components through noun phrase chunking techniques, visualizing components and their relations through WARM and knowledge graph visualizations, and similarity for specifications and requirements with token-based similarity metrics.

At first, we planned on using association rule mining, with rules constituted by components, as a framework to infer similarity in specifications and requirements. However, we realized similarity metrics that better assert component contribution performed better in evaluating the recommendations proposed. We also understood that deep learning performance in NLP tasks does not always correspond to performance in evaluation by engineers.

Therefore, we created a main chunking framework and proposed the adoption of $IoC_{wgt}$, which was validated with regression models. We ran experimentation with four different subsystems, each with a different format and different subdomains of components. We analyzed similarity in two frameworks and obtained 90% of accuracy for some subsystems when using the model to recommend the already existing links in these, and good or very good recommendations in never evaluated input requirements, through engineer validation. Thus, our system can reduce top-K recommendations to a process that only takes a few hours to analyze, depending on the number of input requirements and recommendations to produce.

6.1 Future Work

This work can lead to a series of further research, such as:

- Component extraction without context-free grammars and not requiring external knowledge, or with NER techniques that do not require extensive training sets. It would be pertinent to automate this process since the grammar we define might not produce interesting results with other domains.
• Tweak the NLP pipeline, with the introduction of lemmatization or through the study of the impact of not removing stopwords during the process.

• Study deep learning techniques for NLP tasks that can produce meaningful recommendations to engineers. Lately, state-of-the-art embeddings have been liberated to easy-to-access frameworks. This can lead to more complex architectures that we do not analyze, which could better correspond to engineers’ expectations.

• Execute the recommendation system in other safety-critical domains, and evaluate its performance.
References


# Appendix A

## Rule Mining and Visualizations

Table A.1: Itemsets and rules extracted in $P^2$, with ARM and WARM.

<table>
<thead>
<tr>
<th>D</th>
<th>F</th>
<th>Itemsets</th>
<th>$S_i$</th>
<th>$#T$</th>
<th>Association Rules</th>
<th>$S_r$</th>
<th>$C$</th>
<th>$L$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ARM</strong></td>
<td></td>
<td>(MC)</td>
<td>0.452</td>
<td>85.00</td>
<td>(link voltage) $\Rightarrow$ (DC)</td>
<td>0.150</td>
<td>1.00</td>
<td>2.23</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(DC)</td>
<td>0.445</td>
<td>84.00</td>
<td>(link voltage, MC) $\Rightarrow$ (DC)</td>
<td>0.101</td>
<td>1.00</td>
<td>2.23</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(LB, Converter Box, Separation Contact)</td>
<td>0.031</td>
<td>6.00</td>
<td>(Event, fault reset) $\Rightarrow$ (PC)</td>
<td>0.096</td>
<td>1.00</td>
<td>3.76</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Brake, LC, link voltage)</td>
<td>0.031</td>
<td>6.00</td>
<td>(LC, link voltage) $\Rightarrow$ (DC)</td>
<td>0.090</td>
<td>1.00</td>
<td>2.23</td>
</tr>
<tr>
<td><strong>WARM</strong></td>
<td></td>
<td>(MC)</td>
<td>0.359</td>
<td>67.65</td>
<td>(DC, Brake) $\Rightarrow$ (MC)</td>
<td>0.005</td>
<td>0.79</td>
<td>2.21</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(DC)</td>
<td>0.308</td>
<td>57.94</td>
<td>(Motor Converter, LC) $\Rightarrow$ (MC)</td>
<td>0.013</td>
<td>0.75</td>
<td>2.08</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(AC, Motor Converter)</td>
<td>0.005</td>
<td>96.96</td>
<td>(AC, LC) $\Rightarrow$ (MC)</td>
<td>0.154</td>
<td>50.50</td>
<td>21.11</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(LC, link voltage)</td>
<td>0.005</td>
<td>96.96</td>
<td>(earth fault) $\Rightarrow$ (LC)</td>
<td>0.007</td>
<td>50.50</td>
<td>21.12</td>
</tr>
<tr>
<td><strong>ARM</strong></td>
<td></td>
<td>(MC)</td>
<td>0.267</td>
<td>76.00</td>
<td>(AC, Fault, MC) $\Rightarrow$ (DC)</td>
<td>0.04</td>
<td>1.00</td>
<td>5.68</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(DC)</td>
<td>0.232</td>
<td>66.00</td>
<td>(TDS) $\Rightarrow$ (TCMS)</td>
<td>0.04</td>
<td>1.00</td>
<td>9.79</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(AC, Contactor Feedback, MC)</td>
<td>0.031</td>
<td>0.90</td>
<td>(Converter, DC) $\Rightarrow$ (AC, MC)</td>
<td>0.035</td>
<td>0.50</td>
<td>5.68</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Converter, LC, MC)</td>
<td>0.031</td>
<td>9.00</td>
<td>(DC, DC-link, MC) $\Rightarrow$ (AC, LC)</td>
<td>0.032</td>
<td>0.50</td>
<td>7.47</td>
</tr>
<tr>
<td><strong>WARM</strong></td>
<td></td>
<td>(MC)</td>
<td>0.196</td>
<td>55.88</td>
<td>(Converter, LC, MC) $\Rightarrow$ (AC)</td>
<td>0.006</td>
<td>0.78</td>
<td>3.68</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(DC)</td>
<td>0.129</td>
<td>36.63</td>
<td>(Converter, DC, MC) $\Rightarrow$ (Fault)</td>
<td>0.005</td>
<td>0.75</td>
<td>3.55</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(DC-link, LC)</td>
<td>0.005</td>
<td>1.43</td>
<td>(MC) $\Rightarrow$ (AC)</td>
<td>0.020</td>
<td>0.50</td>
<td>1.87</td>
</tr>
</tbody>
</table>

---

* D: Domain  F: Framework  $S_i$: Itemset Support  $#T$: Number of Transactions  $S_r$: Rule Support  C: Confidence  L: Lift
Table A.2: Itemsets and rules extracted in $P3$, with ARM and WARM.

<table>
<thead>
<tr>
<th>D</th>
<th>F</th>
<th>Itemsets</th>
<th>$S_i$</th>
<th>#T</th>
<th>Association Rules</th>
<th>$S_r$</th>
<th>C</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM</td>
<td>DS</td>
<td>(LC)</td>
<td>0.101</td>
<td>23.00</td>
<td>(main transformer) $\Rightarrow$ (LB)</td>
<td>0.048</td>
<td>0.58</td>
<td>5.97</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(LB)</td>
<td>0.097</td>
<td>22.00</td>
<td>(MC) $\Rightarrow$ (LC)</td>
<td>0.048</td>
<td>0.58</td>
<td>5.71</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>(LC) $\Rightarrow$ (main transformer)</td>
<td>0.048</td>
<td>0.50</td>
<td>5.97</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(DC-link)</td>
<td>0.088</td>
<td>20.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(main transformer)</td>
<td>0.083</td>
<td>19.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WARM</td>
<td>DS</td>
<td>(line voltage)</td>
<td>0.079</td>
<td>17.89</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(DC-link)</td>
<td>0.073</td>
<td>16.67</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(AC)</td>
<td>0.005</td>
<td>1.33</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Motor Converter)</td>
<td>0.005</td>
<td>1.33</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARM</td>
<td>CR</td>
<td>(System)</td>
<td>0.117</td>
<td>8.00</td>
<td>(TCMS) $\Rightarrow$ (System)</td>
<td>0.044</td>
<td>0.60</td>
<td>5.10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(TCMS)</td>
<td>0.073</td>
<td>5.00</td>
<td>(System) $\Rightarrow$ (TCMS)</td>
<td>0.044</td>
<td>0.38</td>
<td>5.11</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(traction motor)</td>
<td>0.044</td>
<td>3.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(TCMS, System)</td>
<td>0.044</td>
<td>3.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WARM</td>
<td>CR</td>
<td>(System)</td>
<td>0.117</td>
<td>7.99</td>
<td>(TCMS) $\Rightarrow$ (System)</td>
<td>0.006</td>
<td>0.59</td>
<td>5.10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(earth fault)</td>
<td>0.027</td>
<td>1.87</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(LC)</td>
<td>0.005</td>
<td>0.40</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(System, TCMS)</td>
<td>0.005</td>
<td>0.40</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D: Domain</th>
<th>F: Framework</th>
<th>S_i: Itemset Support</th>
<th>#T: Number of Transactions</th>
<th>S_r: Rule Support</th>
<th>#T: Number of Transactions</th>
<th>L: Lift</th>
</tr>
</thead>
</table>

Table A.3: Itemsets and rules extracted in $P4$, with ARM and WARM.

<table>
<thead>
<tr>
<th>D</th>
<th>F</th>
<th>Itemsets</th>
<th>$S_i$</th>
<th>#T</th>
<th>Association Rules</th>
<th>$S_r$</th>
<th>C</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM</td>
<td>DS</td>
<td>(DC-link)</td>
<td>0.139</td>
<td>11.00</td>
<td>(GDU) $\Rightarrow$ (DC)</td>
<td>0.038</td>
<td>1.00</td>
<td>26.33</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Motor Converter)</td>
<td>0.114</td>
<td>9.00</td>
<td>(DC) $\Rightarrow$ (GDU)</td>
<td>0.038</td>
<td>1.00</td>
<td>26.33</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(coolant pressure)</td>
<td>0.038</td>
<td>3.00</td>
<td>(IGBT) $\Rightarrow$ (line converter)</td>
<td>0.038</td>
<td>1.00</td>
<td>9.88</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(GDU)</td>
<td>0.038</td>
<td>3.00</td>
<td>(DC-SW) $\Rightarrow$ (MC)</td>
<td>0.038</td>
<td>1.00</td>
<td>8.78</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(DC-link, voltage)</td>
<td>0.139</td>
<td>10.99</td>
<td>(Brake) $\Rightarrow$ (Drive)</td>
<td>0.008</td>
<td>0.33</td>
<td>19.55</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Motor Converter)</td>
<td>0.102</td>
<td>8.10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(IGBT)</td>
<td>0.007</td>
<td>0.60</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Converter Module)</td>
<td>0.005</td>
<td>0.40</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WARM</td>
<td>CR</td>
<td>(Converter)</td>
<td>0.400</td>
<td>48.00</td>
<td>(fault reset signal) $\Rightarrow$ (Converter)</td>
<td>0.150</td>
<td>1.00</td>
<td>2.50</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(DC-link)</td>
<td>0.191</td>
<td>23.00</td>
<td>(DC-link, fault reset signal) $\Rightarrow$ (DC-link)</td>
<td>0.050</td>
<td>1.00</td>
<td>2.50</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Converter, LB)</td>
<td>0.033</td>
<td>4.00</td>
<td>(Converter) $\Rightarrow$ (MC)</td>
<td>0.042</td>
<td>0.50</td>
<td>1.25</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Converter, SBB)</td>
<td>0.033</td>
<td>4.00</td>
<td>(Converter) $\Rightarrow$ (current sensor)</td>
<td>0.042</td>
<td>0.50</td>
<td>2.08</td>
</tr>
<tr>
<td>WARM</td>
<td>CR</td>
<td>(Converter)</td>
<td>0.399</td>
<td>47.90</td>
<td>(coolant pressure) $\Rightarrow$ (Converter)</td>
<td>0.007</td>
<td>0.33</td>
<td>2.08</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(DC-link)</td>
<td>0.115</td>
<td>13.80</td>
<td>(MC) $\Rightarrow$ (Converter)</td>
<td>0.008</td>
<td>0.49</td>
<td>1.25</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(OVC)</td>
<td>0.005</td>
<td>0.60</td>
<td>(MC) $\Rightarrow$ (DC-link)</td>
<td>0.005</td>
<td>0.30</td>
<td>2.61</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Motor contactor)</td>
<td>0.005</td>
<td>0.60</td>
<td>(Converter) $\Rightarrow$ (DC-link)</td>
<td>0.054</td>
<td>0.14</td>
<td>1.19</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D: Domain</th>
<th>F: Framework</th>
<th>S_i: Itemset Support</th>
<th>#T: Number of Transactions</th>
<th>S_r: Rule Support</th>
<th>C: Confidence</th>
<th>L: Lift</th>
</tr>
</thead>
</table>
Figure A.1: Knowledge graphs for DSs and CRs of subsystem $P_2$.

Figure A.2: Knowledge graphs for DSs and CRs of subsystem $P_3$. 
(a) Components in design specifications.  
(b) Components in customer requirements.

Figure A.3: Knowledge graphs for DSs and CRs of subsystem $P^4$. 

88