STANDARDIZATION OF ENGINEERING REQUIREMENTS USING LARGE LANGUAGE MODELS

A Dissertation
Presented to
The Academic Faculty

By

Archana Tikayat Ray

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
Daniel Guggenheim School of Aerospace Engineering
Aerospace Systems Design Laboratory

Georgia Institute of Technology

May 2023

STANDARDIZATION OF ENGINEERING REQUIREMENTS USING LARGE LANGUAGE MODELS

Thesis committee:

Prof. Dimitri N. Mavris, Advisor
School of Aerospace Engineering
Georgia Institute of Technology

Dr. Bjorn F. Cole
Systems Engineering Senior Staff
Lockheed Martin Space Systems

Prof. Daniel P. Schrage
School of Aerospace Engineering
Georgia Institute of Technology

Dr. Olivia J. Pinon Fischer
School of Aerospace Engineering
Georgia Institute of Technology

Dr. Ryan T. White
Department of Mathematical Sciences
Florida Institute of Technology

Date approved: April 13, 2023

For all your days prepare,

And meet them ever alike:

When you are the anvil, bear —

When you are the hammer, strike.

Edwin Markham

To my husband, Anirudh

for being my rock in the storm,

and for making adulting easier.

Couldn't have done this without you...

ACKNOWLEDGMENTS

This dissertation represents the culmination of a journey that has been both intellectually challenging and emotionally rewarding. I have learned to embrace failure as an integral part of the research process, to be persistent in the face of obstacles, and to celebrate small wins. The road to this point has been long, winding, and at times, daunting. But thanks to the support of so many wonderful people, I stand here today, having completed one of the most significant achievements of my life.

First and foremost, I would like to express my deepest gratitude to my advisor, Prof. Dimitri Mavris, whose support, mentorship, and guidance have been instrumental in shaping the person I have become. His encouragement to explore multiple research directions and collaborate with industry sponsors has been instrumental in shaping my research and helping me see the common ground between industry and academia, thus breaking down the silos between the two. I could not have asked for a better advisor.

I would like to express my appreciation to my committee members, Prof. Dimitri Mavris, Prof. Daniel Schrage, Dr. Ryan White, Dr. Bjorn Cole, and Dr. Olivia Pinon Fischer, for their invaluable feedback, guidance, and support throughout this journey. Their expertise, insights, and constructive criticism have been instrumental in shaping my research and helping me grow as a researcher. I am humbled by their dedication and commitment to my professional development.

I would like to extend a heartfelt shoutout to Dr. Olivia Pinon Fischer, who has been an invaluable mentor and friend throughout this journey. Her kindness, generosity, and unwavering encouragement have been a constant source of inspiration and motivation for me. I feel truly fortunate to have had the opportunity to work with such a talented, supportive, and caring individual.

I express my sincere appreciation to Dr. Bjorn Cole for his valuable contribution to

bringing the industry perspective to my dissertation. I am grateful for his willingness to engage in thoughtful discussions and provide insightful opinions whenever needed. Additionally, I am thankful for his exceptional co-authorship on the joint papers we worked on together.

I would like to express my sincere gratitude to Dr. Ryan White, who has been an invaluable guide and friend throughout my undergraduate and graduate studies. I am grateful for his willingness to devote his time and energy to my work, and for the many valuable conversations that we have shared. Thank you, Ryan, for all that you do for me and for the rest of your students.

I would like to extend a special thank you to Adrienne Durham, Tanya Ard-Smith, and Brittany Hodges who have made my life as an international student a little easier. Their invaluable assistance with paperwork, and other administrative tasks was instrumental in ensuring a smooth journey throughout my doctoral program.

To my friends, Rubanya Nanda, Lijing Zhai (and Sunny), Patsy Jammal, Stella Kampezidou, Nathaniel Omoarebun, Rosa Galindo, and Shubhneet Singh who have been my pillars of strength, my sounding board, and my cheerleaders throughout this journey – thank you. Your support and encouragement have helped me through the most challenging times, and I am grateful for the laughs, tears, and memories we've shared along the way.

I am also immensely grateful to my family, who have been my source of support, love, and encouragement. Their constant belief in me, even when I doubted myself, has meant the world to me. They have shared the ups and downs of this journey, and I know that I could not have made it without them.

Last but certainly not least, I want to express my heartfelt gratitude to my husband Anirudh Bhat, who has been my ardent supporter and partner throughout this journey. His countless sacrifices, whether it was staying up late into the night discussing ideas or helping me debug my code, have been instrumental in making this

achievement possible. I could not have done this without him, and I am forever grateful for his presence in my life. I eagerly look forward to embarking on the next chapter of our lives together.

Archana Tikayat Ray

April 13, 2023

TABLE OF CONTENTS

Acknowledgments v

List of Tables xiii

List of Figures xvi

List of Acronyms xxii

Summary xxvi

Chapter 1: Introduction 1

1.1 Integrated Product and Process Development (IPPD) 1

1.2 Quality Function Deployment (QFD) 3

1.3 The Systems Engineering Process 5

1.4 Requirements: Background and definitions 7

1.4.1 Requirements Engineering 7

1.5 Cost of bad requirements 9

1.6 Challenges with Natural Language (NL) Requirements and proposed solution 11

1.7 Summary 14

1.8 Dissertation Overview 16

1.9 Publications 18

Chapter 2: Preliminaries and Literature Review 19

2.1 Natural Language Processing for Requirements Engineering (NLP4RE) 19

2.2 NLP and Case for Aerospace specific corpus 22

2.2.1 Natural Language Processing (NLP) 22

2.2.2 Language Model (LM) and corpus 24

2.2.3 Importance of domain-specific corpus 27

2.3 Identification of named entities (NEs) in aerospace requirements . . . 28

2.4 Classification of aerospace requirements 30

2.5 Standardization of aerospace requirements and use of boilerplates . . 34

2.6 Summary of Observations and Gaps 38

Chapter 3: Research Formulation 40

3.1 Research Objective 40

3.2 Research Question 1: Incorporating aerospace domain knowledge into LMs 42

3.3 Research Question 2: Identification of boilerplates 59

3.4 Overarching Hypothesis 70

3.5 Summary of Observations, gaps, and research questions 71

Chapter 4: Methodology 73

4.1 Step 0: Development of an annotated aerospace corpus 74

4.2 Step 1: Fine-tuning BERT for aerospace NER (aeroBERT-NER) . . . 75

4.3 Step 2: Fine-tuning BERT for aerospace requirement classification (aeroBERT-Classifier) 76

4.4 Step 3: Fine-tuning BERT for POS tagging of aerospace text (aeroBERT-POStagger) 78

4.5 Step 4: Creation of aerospace requirements boilerplates 79

Chapter 5: Implementation 80

5.1 Creation of annotated aerospace corpora 80

5.1.1 Annotated corpus for aerospace Named-entity recognition (NER) 80

5.1.2 Annotated corpus for aerospace requirements classification . . 85

5.2 Fine-tuning BERT for aerospace NER (aeroBERT-NER) 88

5.2.1 Preparing the dataset for fine-tuning BERT for aerospace NER 88

5.2.2 Fine-tuning BERT for aerospace NER 90

5.3 Fine-tuning BERT for aerospace requirements classification (aeroBERT-Classifier) 94

5.3.1 Preparing the dataset for fine-tuning BERT for classification of aerospace requirements 94

5.3.2 Fine-Tuning BERT LM for aerospace requirements classification 97

5.4 Fine-tuning BERT for POS tagging of aerospace text 100

5.4.1 Observations regarding POS tags and patterns in requirements 102

5.5 Standardization of requirements 110

5.5.1 Requirements Table 111

5.5.2 Requirements Boilerplate 113

5.5.3 Identification of boilerplates by examining patterns in requirements text 118

Chapter 6: Results and Discussion 120

6.1 Creation of annotated aerospace corpora 120

6.2 aeroBERT-NER 121

6.2.1 Model performance vs. dataset size 121

6.2.2 Glossary creation 125

6.2.3 Comparison between aeroBERT-NER and $\text{BERT}_{\text{BASE}}$-NER . . 127

6.3 aeroBERT-Classifier 128

6.3.1 aeroBERT-Classifier Performance 128

6.3.2 Comparison between aeroBERT-Classifier and other text classification LMs 132

6.4 Standardization of requirements 134

6.4.1 Creation of requirements table 134

6.4.2 Identification of boilerplates by examining patterns in requirements text 136

Chapter 7: Practitioner's Guide 151

7.1 Creation of aeroBERT-NER and aeroBERT-Classifier 151

7.2 Creation of Requirements Table using language models 154

7.3 Identification of Requirements boilerplates using language models . . 156

Chapter 8: Concluding Remarks 159

8.1 Conclusions 159

8.2 Summary of Contributions 161

8.3 Limitations and Recommendations for Future Work 163

8.3.1 aeroBERT-NER 163

8.3.2 aeroBERT-Classifier 164

8.3.3 Standardization of requirements 165

Appendices 166

Appendix A: Datasets 167

Appendix B: Test set for identification of named entities 169

Appendix C: Definitions and NLP Concepts 171

References 180

LIST OF TABLES

1.1 Order of requirements engineering 9

1.2 Cost to fix requirements error 11

2.1 Requirements examples from the PROMISE NFR dataset 32

3.1 Criteria considered when choosing a LM 44

3.2 Named Entity Recognition by $BERT_{BASE}$ 52

3.3 CoNLL-2003 Language-Independent Named Entity Recognition dataset 52

3.4 BIO tagging notation 55

3.5 BIO tagging NE annotation task 56

3.6 Dataset for requirements classification 62

3.7 POS tagging notation used by Flair 64

4.1 Overview of Proposed Methodology 73

4.2 Title 14 CFR Parts 74

5.1 Resources for creation of annotated aerospace NER corpus 81

5.2 Text modification and requirement creation 82

5.3 Symbols that were modified for corpus creation 82

5.4 Types of named-entities identified 83

5.5 NER tags and their counts in the aerospace corpus for aeroBERT-NER 84

5.6 Resources used for the creation of aerospace requirements classification corpus 85

5.7 Definitions used for labeling/annotating requirements [5], [7], [121] . . 86

5.8 Examples showing the modification of Interface requirements into other "types" of requirements 87

5.9 Requirement classification dataset format 88

5.10 NER tags and their corresponding IDs 91

5.11 Tokens, token ids, tags, tag ids, and attention masks 92

5.12 Breakdown of the "types" of requirements in the training and test set 94

5.13 Tokens, token IDs, and attention masks for requirements classification 96

5.14 List of language models used for benchmarking the performance of aeroBERT-Classifier 100

5.15 Text chunks: definitions and examples 105

5.16 List of language models used to populate the columns of requirement table 114

5.17 Elements of requirement boilerplate templates and their definitions . 116

6.1 Trends in F1 scores with the increase in dataset size 123

6.2 Model performance for aeroBERT-NER on the validation set 124

6.3 Percentage of NEs identified by aeroBERT-NER in the test set 126

6.4 Example of NEs identified and added to the glossary 126

6.5 Comparison of aeroBERT-NER with $\text{BERT}_{\text{BASE}}$-NER 127

6.6 Model performance for aeroBERT-Classifier on the test set 129

6.7 List of requirements (from test set) that were misclassified (0: Design; 1: Functional; 2: Performance) 131

6.8 Requirement table populated by using various LMs 135

6.9 Summary of boilerplate template identification task 136

B.1 Test set containing 20 aerospace requirements 169

LIST OF FIGURES

1.1 Integrated Product and Process Development (IPPD) 2

1.2 House of Quality - Design tool for QFD 4

1.3 The Systems Engineering Process 6

1.4 The Systems Engineering Engine 8

1.5 Information-based Requirement Development and Management Model 14

1.6 Semi-machine-readable requirement: data objects 16

1.7 Phases of requirements engineering: NL requirements to models . . . 16

2.1 Timeline of published studies in NLP4RE 20

2.2 Distribution of selected studies based on NLP4RE task 21

2.3 NLP and its subfields 23

2.4 Problems with NL requirements and potential solutions 24

2.5 Language Model and its types 25

2.6 Chronology of Corpus, Language Model, and downstream tasks . . . 26

2.7 NLP pipeline for text chunking and NER 28

2.8 Pipeline for classifying NFRs using ML 31

2.9 Rupp's Boilerplate 35

2.10 EARS Boilerplate 36

2.11 Majo and Jaramillo template 37

3.1 Research road-map 42

3.2 Example explaining the meaning of bidirectional 45

3.3 Transformer Architecture illustrating language translation task 45

3.4 Transformer detailed architecture 46

3.5 $\text{BERT}_{\text{BASE}}$ and $\text{BERT}_{\text{LARGE}}$ 47

3.6 Pre-training and fine-tuning BERT language model 48

3.7 BERT attention head illustration 49

3.8 BERT embeddings and use of special tokens 51

3.9 Pre-training and fine-tuning BERT LM 51

3.10 $\text{BERT}_{\text{BASE}}$ NER language model applied to general-domain and aerospace texts 53

3.11 Using transfer learning to fine-tune LMs for tasks in the aerospace domain 54

3.12 Confusion Matrix 58

3.13 Research Question 2 flowchart 60

3.14 Example showing POS tagging by Flair 64

3.15 Example showing POS tags 65

3.16 Example showing words with POS tags based on context 65

3.17 Boilerplate identification using linguistic constructs 68

3.18 Summary of Observations, gaps, and research questions 72

4.1 Overview of Proposed Methodology 73

4.2 Methodology for obtaining aeroBERT-NER 76

4.3 Methodology for obtaining aeroBERT-Classifier 77

4.4 aeroBERT-classifier flowchart for requirement classification 77

4.5 Methodology for obtaining aeroBERT-POStagger 78

4.6 Methodology for aerospace requirement boilerplate construction . . . 79

5.1 Types of sentences included in the aerospace corpus 81

5.2 Flowchart showing creation of lookup files for NER annotation 83

5.3 Flowchart showing NER annotation logic 84

5.4 Types of requirements considered for the requirements classification task 87

5.5 Choosing maximum length of the input sequence for training aeroBERT-NER 90

5.6 Detailed methodology for full-fine-tuning of BERT LM for obtaining aeroBERT-NER 93

5.7 Figure showing the distribution of sequence lengths in the training set used for training aeroBERT-Classifier 95

5.8 Demonstration of batch size for classification 97

5.9 Detailed methodology used for full-fine-tuning of BERT for aerospace requirements classification 98

5.10 Demonstration of intraclass variations in POS tags 101

5.11 Sankey diagram showing the POS tag patterns in design requirements. 102

5.12 Sankey diagram showing the POS tag patterns in functional requirements 103

5.13 Sankey diagram showing the POS tag patterns in performance requirements 103

5.14 Demonstration of sentence chunks 104

5.15 Sankey diagram showing the text chunk patterns in design requirements 106

5.16 Text chunk analysis for design requirements (Part 1) 107

5.17 Text chunk analysis for design requirements (Part 2) 107

5.18 Sankey diagram showing the text chunk patterns in functional requirements 108

5.19 Text chunk analysis for functional requirements 109

5.20 Sankey diagram showing the text chunk patterns in performance requirements 109

5.21 Text chunk analysis for performance requirements 110

5.22 Pipeline for converting NL requirements to standardized requirements using various LMs 111

5.23 SysML requirements table 112

5.24 Requirement boilerplate identification pipeline 114

5.25 Example demonstrating the boilerplate identification pipeline 115

5.26 SysML requirements table 115

6.1 Comparing validation performance of aeroBERT-NER when trained and tested on different dataset sizes 122

6.2 Cased vs. Uncased text 123

6.3 Confusion matrix showing the breakdown of the true and predicted labels by the aeroBERT-Classifier on the test data 131

6.4 Confusion matrix showing the breakdown of the true and predicted labels by the bart-large-mnli model on the test data 133

6.5 Design Requirements: Boilerplate 1 138

6.6 Design Requirements: Boilerplate 2 139

6.7 Functional Requirements: Boilerplate 1 141

6.8 Functional Requirements: Boilerplate 2 142

6.9 Functional Requirements: Boilerplate 3 143

6.10 Functional Requirements: Boilerplate 4 145

6.11 Functional Requirements: Boilerplate 5 146

6.12 Performance Requirements: Boilerplate 1 147

6.13 Performance Requirements: Boilerplate 2 148

6.14 Performance Requirements: Boilerplate 3 149

7.1 Practitioner's Guide to creation of aeroBERT-NER and aeroBERT-Classifier 152

7.2 Practitioner's Guide to the creation of requirements table 155

7.3 Practitioner's Guide to creation of aeroBERT-NER and aeroBERT-Classifier 157

8.1 Future Work: Overlapping NEs example 163

C.1 Lemmatization 171

C.2 Dependency Parsing 171

C.3 Word sense disambiguation - A computer mouse vs. a mouse (rodent) 172

C.4 Cased and uncased text for BERT language model 172

C.5 NER example 172

C.6 Tokenization example 172

C.7 WordPiece Tokenizer 173

C.8 Stop words 173

C.9 Word Embeddings in vector space 173

C.10 Word Embeddings Word2Vec 174

C.11 BERT Embeddings 174

C.12 BERT Embeddings deep-dive 174

C.13 Transformer Architecture 175

C.14 Query, Key, and Value matrices 177

C.15 Example showing Query (Q) and Key (K) vectors and attention score calculation - Part 1 178

C.16 Example showing Query (Q) and Key (K) vectors and attention score calculation - Part 2 178

C.17 Matrix multiplication intuition for the calculation of multi-headed attention 179

LIST OF ACRONYMS

ADJP Adjective Clause

ADVP Adverbial Clause

AoA Angle-of-Attack

ATC Air Traffic Control

BART Bidirectional Auto-Regressive Transformers

BERT Bidirectional Encoder Representations from Transformers

CD Cardinal Numbers

CFR Code of Federal Regulations

COND Condition

CNN Convolutional Neural Networks

DATETIME Date time

DET Determiner

EARS Easy Approach to Requirements Syntax

ETOPS Extended Operations

FAA Federal Aviation Administration

FAR Federal Aviation Regulations

FARS Fatality Analysis Reporting System

FinBERT Financial BERT

FN False Negative

FP False Positive

FR Functional Requirements

FUNC Function

GPT-2 Generative Pre-trained Transformer 2

GPT-3 Generative Pre-trained Transformer 3

GPU Graphics Processing Units

HMM Hidden Markov Models

INCOSE International Council on Systems Engineering

I-NRDM Information-Based Needs and Requirements Definition and Management

IPPD Integrated Product & Process Design

K Key

LLM Large Language Model

LM Language Model

LOC Location

LSRE Large Scale Requirements Engineering

LSTM Long Short-Term Memory

MBSE Model-Based Systems Engineering

MISC Miscellaneous

ML Machine Learning

MLM Masked Language Modeling

MSRE Medium-Scale Requirements Engineering

NASA National Aeronautics and Space Administration

NE Named Entities

NER Named-Entity Recognition

NFR Non-Functional Requirements

NL Natural Language

NLG Natural Language Generation

NLI Natural Language Inference

NLP Natural Language Processing

NLP4RE Natural Language Processing for Requirements Engineering

NLTK Natural Language Toolkit

NLU Natural Language Understanding

NN Neural Network

NN Noun

NoRBERT Non-functional and functional Requirements classification using BERT

NP Noun Phrases

NSP Next Sentence Prediction

OEM Original Equipment Manufacturer

ORG Organizations

PER Person

POS Parts-Of-Speech

PP Prepositional Phrase

Q Query

QA Question Answering

QFD Quality Function Deployment

RES Resource

RNN Recurrent Neural Network

RQ Research Question

RS Requirement Specification

SADT Structured Analysis Design Technique

SBAR Subordinate Clause

SDM System Design Methodology

SDMC Sequential Data Mining under Constraints

SL Supervised Learning

SME Subject Matter Expert

SOTA State-Of-The-Art

SRS Software Requirement Specification

SSRE Small-Scale Requirements Engineering

SVM Support Vector Machines

SYS System

SysML Systems Modeling Language

T5 Text-to-Text Transfer Transformer

TN True Negative

TP True Positive

UML Unified Modeling Language

V Value

VAL Value

VLSRE Very Large-Scale Requirements Engineering

VP Verb Phrases

ZSL Zero-Shot Learning

SUMMARY

Requirements serve as the foundation for all systems, products, services, and enterprises. A well-formulated requirement conveys information, which must be necessary, clear, traceable, verifiable, and complete to respective stakeholders. Various types of requirements like functional, non-functional, design, quality, performance, and certification requirements are used to define system functions/objectives based on the domain of interest and the system being designed.

Organizations predominantly use natural language (NL) for requirements elicitation since it is easy to understand and use by stakeholders with varying levels of experience. In addition, NL lowers the barrier to entry when compared to model-based languages such as Unified Modeling Language (UML) and Systems Modeling Language (SysML), which require training. Despite these advantages, NL requirements bring along many drawbacks such as ambiguities associated with language, a tedious and error-prone manual examination process, difficulties associated with verifying requirements completeness, and failure to recognize and use technical terms effectively. While the drawbacks associated with using NL for requirements engineering are not limited to a single domain or industry, the focus of this dissertation will be on aerospace requirements.

Most of the systems in the present-day world are complex and warrant an integrated and holistic approach to their development to capture the numerous interrelationships. To address this need, there has been a paradigm shift towards a model-centric approach to engineering as compared to traditional document-based methods. The promise shown by the model-centric approach is huge, however, the conversion of NL requirements into models is hindered by the ambiguities and inconsistencies in NL requirements. This necessitates the use of standardized/semi-machine-readable requirements for transitioning to Model-Based Systems Engineering (MBSE).

As such, the objective of this dissertation is to identify, develop, and implement tools and techniques to enable/support the automated translation of NL requirements into semi-machine-readable requirements. This will contribute to the mainstream adoption of MBSE.

Given the close relationship between NL and requirements, researchers have been striving to develop Natural Language Processing (NLP) tools and methodologies for processing and managing requirements since the 1970s. Despite the interest in using NLP for requirements engineering, the inadequate developments in language processing technologies thwarted progress. However, the recent developments in this field have propelled NLP for Requirements Engineering (NLP4RE) into an active area of research. Hence, NLP techniques are strong candidates for the standardization of NL requirements and are the focus of this dissertation.

One of the central ideas in NLP is neural language models (LMs), which leverage neural networks to simultaneously learn lower-dimensional word embeddings and learn to estimate conditional probabilities of the next words simultaneously using gradient-based supervised learning. This opened the door to ever-more-complex and effective language models to perform an expanding array of NLP tasks, starting with distinct word embeddings to recurrent neural networks (RNNs) and LSTM encoder-decoders to attention mechanisms. These models did not stray too far from the N-gram statistical language modeling paradigm, with advances that allowed text generation beyond a single next word with for example beam search and sequence-to-sequence learning. These ideas can be applied to distinct NLP tasks. In 2017, the Transformer architecture was introduced which improved computational parallelization capabilities over recurrent models and therefore enabled the successful optimization of larger models. Transformers consist of stacks of encoders (encoder block) and stacks of decoders (decoder block), where the encoder block receives the input from the user and outputs a matrix representation of the input text. The decoder takes the

input representation produced by the encoder stack and generates outputs iteratively.

BERT, which is a transformer-based model was selected for this research because 1) it can be fine-tuned for a variety of language tasks such as Named-entity recognition (NER), parts-of-speech (POS) tagging, and sentence classification, 2) can achieve State-of-the-art (SOTA) results. In addition, it uses a bidirectional transformer-based architecture enabling it to better capture the context in a sentence. BERT is pre-trained on BookCorpus and English Wikipedia (general-domain text) and as a result, needs to be fine-tuned using an aerospace corpus to be able to generalize to the aerospace domain.

To fine-tune BERT for different NLP tasks, two annotated aerospace corpora were created. These corpora contain text from Parts 23 and 25 of Title 14 of the Code of Federal Regulations (CFRs) and publications by the National Academy of Space Studies Board. Both corpora were open-sourced to make them available to other researchers to accelerate research in the field of Natural Language Processing for Requirement Engineering (NLP4RE).

First, the corpus annotated for aerospace-specific named entities (NEs), was used to fine-tune different variants of the BERT LM for the identification of five categories of named entities, namely, system names (SYS), resources (RES), values (VAL), organization names (ORG), and datetime (DATETIME). The extracted named entities were used to create a glossary, which is expected to improve the quality and understandability of aerospace requirements by ensuring uniform use of terminologies. Second, the corpus annotated for aerospace requirements classification was used to fine-tune BERT LM to classify requirements into different types such as design requirements, functional requirements, and performance requirements. Being able to classify requirements will improve the ability to conduct redundancy checks, evaluate consistency, and identify boilerplates, which are pre-defined linguistic patterns for standardizing requirements. Third, an off-the-shelf model (`flair/chunk-english`)

was used for identifying the different sentence chunks in a requirement sentence, which is helpful for ordering phrases in a sentence and hence useful for the standardization of requirements.

The capability to classify requirements, identify named entities occurring in requirements, and extract different sentence chunks in aerospace requirements, facilitated the creation of requirements table and boilerplates for the conversion of NL requirements into semi-machine-readable requirements. Based on the frequency of different linguistic patterns, boilerplates were constructed for various types of requirements.

In summary, this effort resulted in the development of the first open-source annotated aerospace corpora along with two LMs (aeroBERT-NER, and aeroBERT-Classifier). Various methodologies were developed to use the fine-tuned LMs to standardize requirements by making use of requirements boilerplates. As a result, this research will lead to speeding up the design and development process by reducing ambiguities and inconsistencies associated with requirements. In addition, it will reduce the workload on engineers who manually evaluate a large number of requirements by facilitating the conversion of NL aerospace requirements into standardized requirements.

CHAPTER 1
INTRODUCTION

This chapter serves as an introductory section that covers the fundamental concepts of Integrated Product and Process Development (IPPD), the Systems engineering process, and Quality Function Deployment (QFD). It highlights the criticality of requirements engineering in the design of systems, products, and enterprises. Additionally, it underscores the challenges and constraints associated with the use of natural language (NL) for requirements elicitation. Furthermore, it explores the advantages of adopting model-based methodologies and evaluates the obstacles such a shift faces because of the inherent ambiguities in NL requirements. The chapter culminates with a brief summary of the dissertation's main focus.

In the next few sections the Integrated Product and Process Development (IPPD) process, Quality Function Deployment (QFD), and Systems Engineering Process will be discussed. These are interconnected and complementary processes in product development.

1.1 Integrated Product and Process Development (IPPD)

Organizations use the Integrated Product and Process Development (IPPD) [1] method to ensure that the development and production functions, along with the Systems Engineering Process and Quality Function Deployment (QFD), are integrated to create a high-quality product that satisfies customer demands while minimizing development time and expenses. IPPD is utilized to achieve cost, schedule, and quality objectives while designing and developing products that meet customer requirements. The process involves the collaboration of all stakeholders, including customers, suppliers, and team members, to develop a product that satisfies all stakeholders. Figure 1.1 shows

the IPPD process in detail.

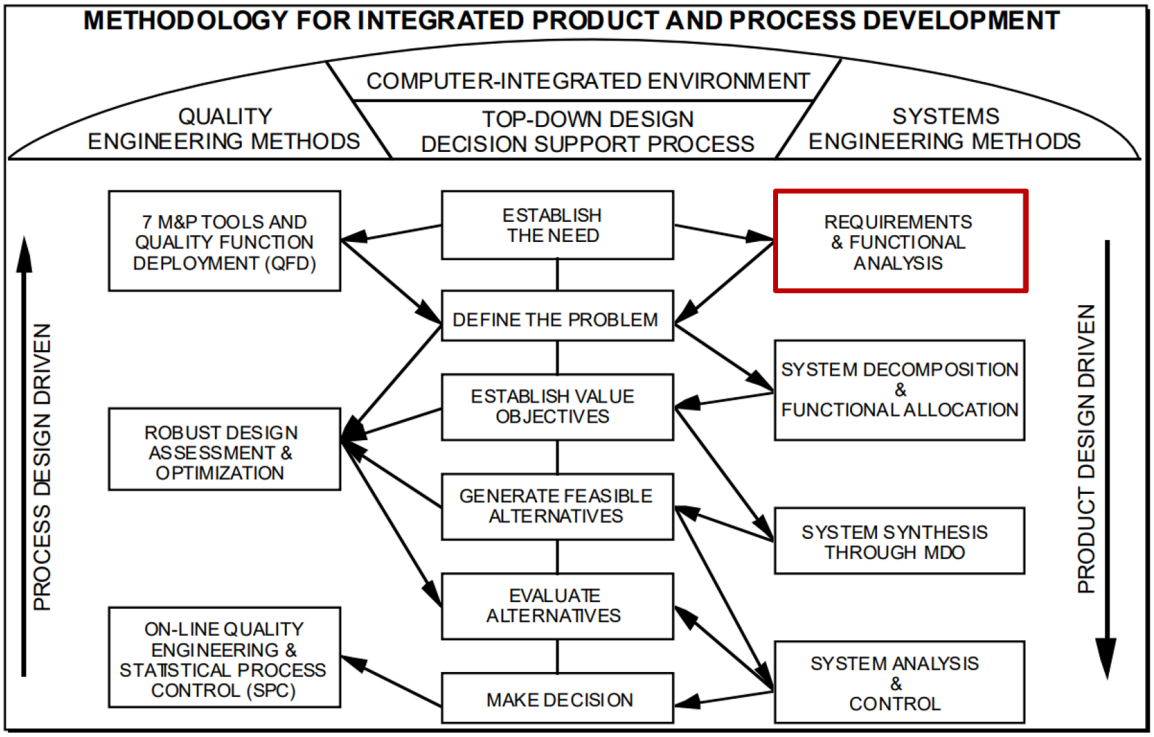


Figure 1.1: Integrated Product and Process Development (IPPD)[1]. The primary focus of this dissertation is on the *Requirements and Functional Analysis* phase of the IPPD process within the systems engineering domain (as highlighted in red the figure).

The IPPD process commonly consists of the following steps:

1. **Identify Customer Requirements:** The first step is to identify customer requirements, including functional and non-functional requirements, quality standards, and regulatory requirements.

2. **Define Objectives:** Once customer requirements are identified, the team must define objectives for the product, including cost, schedule, and quality objectives.

3. **Design and Develop the Product:** The next step is to design and develop the product, taking into account the customer requirements and the defined objectives.

4. **Test and Validate:** After the product is designed and developed, it must be tested and validated to ensure that it meets customer requirements and the defined objectives.

5. **Continuous Improvement:** Finally, the team must continuously improve the product and the process to ensure that the product remains competitive and meets customer needs.

The emphasis of this dissertation is on the *Requirements and Functional Analysis* stage of the IPPD process in the domain of systems engineering. It investigates diverse NLP techniques to accelerate the process of writing requirements and assist in achieving consistency and standardization in NL requirements, which frequently contain ambiguities and inconsistencies that may impede and prolong downstream processes.

1.2 Quality Function Deployment (QFD)

Quality Function Deployment (QFD) (Figure 1.2) is a methodology used in the IPPD process to ensure that customer requirements are translated into the design of the product. QFD uses a matrix to organize and prioritize customer requirements and to link them to specific design characteristics of the product.

The QFD matrix commonly consists of the following components:

1. **Customer Requirements:** This is the first row of the matrix, which lists all the requirements that the customer has for the product.

2. **Importance Rating:** This is the second row of the matrix, which assigns a weight or importance rating to each customer requirement. The importance rating is usually based on how critical the requirement is to the customer.

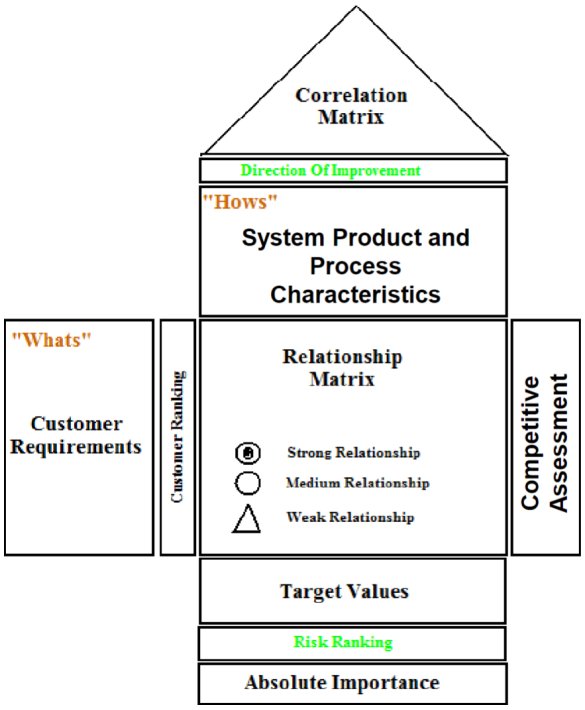


Figure 1.2: House of Quality which is a design tool for QFD [2]

3. **Design Characteristics:** This is the third row of the matrix, which lists all the design characteristics that must be addressed to meet the customer requirements.

4. **Relationship Matrix:** This is the main body of the matrix, which links the customer requirements to the design characteristics. The relationship matrix shows how well each design characteristic satisfies each customer requirement.

5. **Technical Response:** This is the last row of the matrix, which shows how well the design characteristics are being met by the technical response.

The QFD matrix helps to ensure that the design of the product is aligned with the customer's requirements and that the product meets the defined objectives for cost, schedule, and quality.

1.3 The Systems Engineering Process

The systems engineering process is a top-down comprehensive, iterative, and recursive problem-solving process, applied sequentially through all stages of development, that is used to [3]:

- Transform needs and requirements into a set of system product and process descriptions (adding value and more detail with each level of development),
- Generate information for decision-makers, and
- Provide input for the next level of development

The cornerstone activities of systems engineering include Requirements Analysis, Functional Analysis and Allocation, and Design Synthesis (Figure 1.3). These are complemented by a set of techniques and tools collectively known as System Analysis and Control, which aim to monitor decisions and requirements, establish and maintain technical baselines, manage interfaces and risks, track cost and schedule, monitor technical performance, ensure requirements are met through verification, and review and audit progress [3].

This dissertation focuses on the *Requirements Analysis* phase, with reference to the overall *Requirements Loop*. During this phase, functional and performance requirements are generated by interpreting customer requirements and specifying the system's functions and performance level. Systems engineers must ensure that the requirements are clear, concise, comprehensive, unambiguous, and easily understandable, especially since they are typically expressed in Natural Language (NL). To aid in this process, natural language processing (NLP) can be employed to guarantee that requirements meet these criteria.

To summarize, requirements are at the basis of QFD, the Systems Engineering process, and the IPPD process that integrates all of the development and production

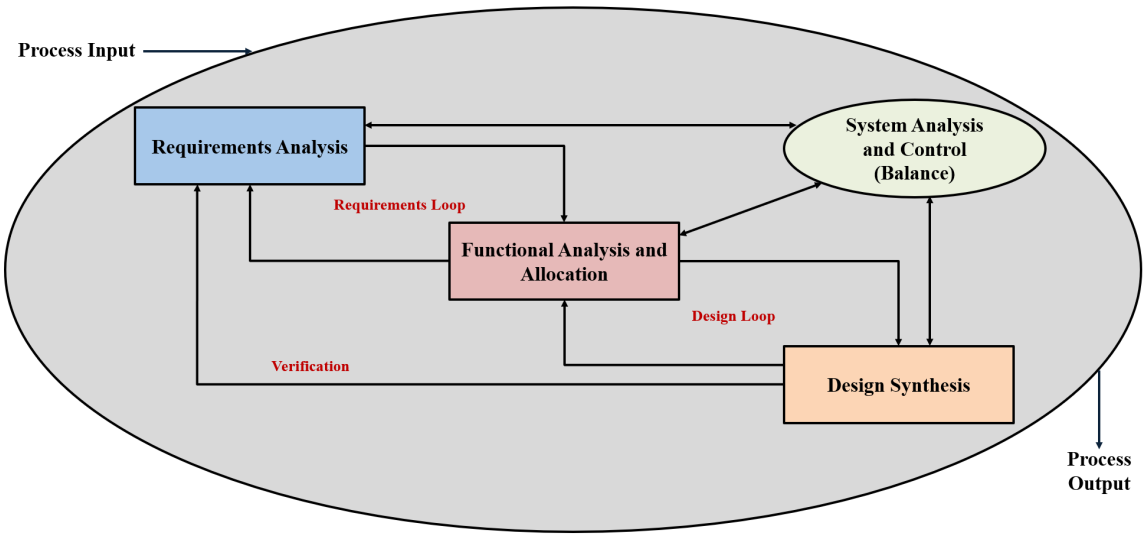


Figure 1.3: The Systems Engineering Process [3]. This dissertation centers around the initial phase of systems engineering, known as *Requirements Analysis*, with some emphasis on the *Requirements Loop* as a whole.

functions. In QFD, requirements are gathered from customers and translated into specific engineering requirements to ensure that the final product meets the customers' needs. In the Systems Engineering Process, requirements analysis is the first step, and it involves developing functional and performance requirements that define what the system must do and how well it must perform. The IPPD process requires integrated development, which means that all life cycle needs must be considered concurrently during the development process, and requirements analysis is a critical component of this process. In all three methodologies, requirements serve as a guide for the design, development, and testing of a product or system and ensure that it meets the intended purpose and specifications.

The remainder of this dissertation will concentrate on requirements, specifically addressing challenges that may arise in natural language (NL) requirements and exploring how modern NLP techniques can mitigate these challenges.

1.4 Requirements: Background and definitions

Requirements serve as the foundation for all systems, products, services, and enterprises. INCOSE [4] defines a requirement as "a statement that identifies a system, product or process characteristic or constraint, which is unambiguous, clear, unique, consistent, stand-alone (not grouped), and verifiable, and is deemed necessary for stakeholder acceptability." Requirements shall be [5], [6]:

- **Necessary:** capable of conveying what is necessary to achieve the required system functionalities, while being compliant with regulations
- **Clear:** able to convey the desired goal to the stakeholders by being simple and concise
- **Traceable:** able to be traced back to higher-level specifications, and vice versa
- **Verifiable:** can be verified by making use of different verification processes, such as analysis, inspection, demonstration, and test
- **Complete:** the requirements should result in a system that successfully achieves the client's needs, while being compliant with the regulatory standards

An example of a "good" requirement is, *"The air-taxi shall have a configuration that can seat five passengers within its passenger cabin."* As formulated, this requirement is necessary, clear, and verifiable. This being a single requirement, one cannot comment on the traceability and completeness properties of a system's set of requirements for this example.

1.4.1 Requirements Engineering

According to NASA Systems Engineering Handbook, requirements engineering is the very first step in the System Design Process (Figure 1.4), which involves defining

the stakeholder expectations and then converting these expectations into technical requirements [7]. In other words, it involves defining, documenting, and maintaining requirements throughout the engineering lifecycle [8]. External stakeholders that contribute to the requirements generation process are competitors, regulatory authorities, operators, shareholders, subcontractors, component providers, consumers, etc. Some of the internal stakeholders are technology research and development teams, sourcing teams, supply and manufacturing teams, engineers, etc. In the case of large-scale systems, the number of stakeholders increases and so do the number of requirements to achieve the desired system [9]. The process followed for defining, documenting, and maintaining requirements is called **requirements engineering** [8]. Requirements engineering can be classified into different categories based on the number of requirements, as shown in Table 1.1 [9].

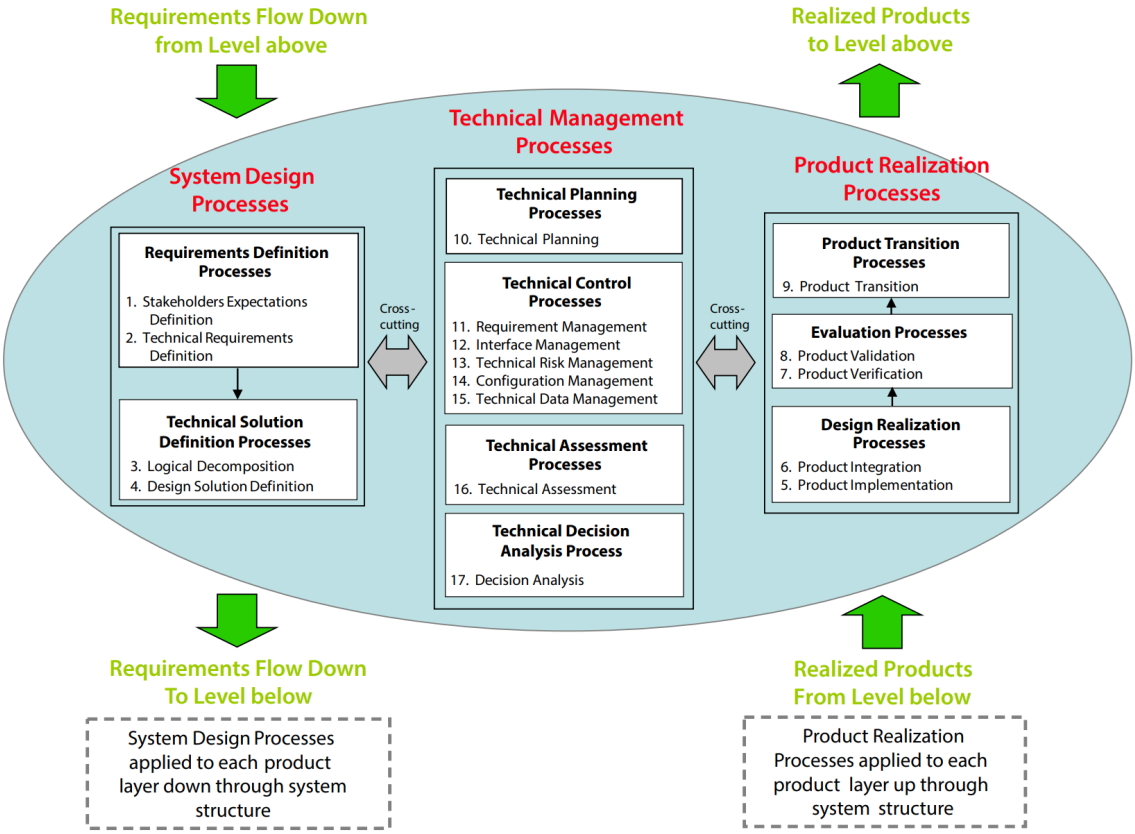


Figure 1.4: The Systems Engineering Engine (NPR 7123.1) [7]

The number of requirements serves as a proxy for the system complexity. Hence, the higher the number of requirements, the higher the system complexity.

Table 1.1: Order of requirements engineering [9]

Level	Number of requirements
Small-Scale Requirements Engineering (SSRE)	$\sim$ 10 requirements
Medium-Scale Requirements Engineering (MSRE)	$\sim$ 100 requirements
Large-Scale Requirements Engineering (LSRE)	$\sim$ 1000 requirements
Very Large-Scale Requirements Engineering (VLSRE)	$\sim$ 10000 requirements

In the aerospace engineering domain, most systems require Very Large-Scale Requirements Engineering (VLSRE) where system requirements are predominantly written in natural language[1] [9], [11], [12]. This is done to make sure that the requirements are easy to write and understand by stakeholders with varying levels of experience when it comes to requirements engineering [11]. However, the use of natural language for requirements engineering might introduce ambiguities, and inconsistencies that would reduce system quality, and lead to cost overruns, and system failure altogether [13].

Observation 1

Requirements are written in Natural Language (NL) to make them more accessible to different stakeholders; however, this introduces unintended inconsistencies and ambiguities.

1.5 Cost of bad requirements

Well-defined requirements and good requirements engineering practices are critical to the successful design, development, and operation of systems, products, and processes. Errors during the requirement definition phase, on the other hand, can trickle down to downstream tasks such as system architecting, system design, implementa-

[1] Any language that has evolved via repetition through use by humans over the years [10] – Example: spoken and written Odia, Hindi, English etc.

tion, inspection, and testing [14], leading to dramatic engineering and programmatic consequences when caught late in the product life cycle [13], [15]. When "requirements engineering" is practiced as a separate effort on large teams, typically with a dedicated team or at least a lead, it becomes very process-focused. Configuration management, customer interviews, validation and verification planning, and maturation sessions all take place within the effort. Specialized software packages such as DOORS [16] have been crafted over many years to service the needs of processing requirements as a collection of individual data records. However, one connection that is often lost is that between the requirements development team and the architectural team. Because Natural Language (NL) is predominantly used to write requirements [11], requirements are commonly prone to ambiguities and inconsistencies. This in turn increases the likelihood of errors and issues in the way requirements are formulated. The ambiguities of the requirements text are eventually resolved, and not entirely satisfactorily, in test definition. At this point, misunderstandings and misses are quite expensive. If the misses don't prevent the system from being fielded, they will often be overlooked and instead simply become potentially lost value to the end user. This overall process orientation can make requirements development look like it is just part of the paperwork overhead of delivering large projects to institutional customers rather than the vital part of customer needs understanding and embodiment that it is.

According to a study by NASA [15], the cost to fix requirements during the requirements generation phase is minimal but can go up to 29x – 1500x in the operations phase (Table 1.2). According to data from industry, 50% of the product defects and 80% of rework effort can be traced back to the errors during the requirements engineering phase [14]. The stakes are even higher when it comes to safety-critical systems[2] – 40% of accidents involving these systems have resulted due to poor requirements [14].

[2]Systems whose failure can lead to catastrophic damage to life, property, and environment. Examples: medical devices, nuclear power plant systems, aircraft flight control systems [17]

This emphasizes the importance of requirements and how it is even more important to fix errors at an early stage to save time and costs – which consequently means allocating a larger amount of project costs towards the requirement definition phase [15].

Table 1.2: Cost to fix requirements error at NASA (in ratios) [15]

Development Phase	Cost to fix
Requirements generation phase	1 (baseline)
Design phase	3x – 8x
Manufacturing/Build phase	7x – 16x
Integration/Test phase	21x – 78x
Operations phase	29x – 1500x

Issues with requirements traditionally include poor requirements quality, inappropriate constraints, non-traceability of requirements, missing requirements (usually the non-functional requirements since it is assumed that they are obvious – availability, interoperability, performance, reliability, robustness, safety, security, stability, usability, etc.), inadequate verification of requirements quality, inadequate requirements validation and management, etc. [14].

Observation 2

The cost of fixing errors in requirements goes up exponentially as we progress across the project life cycle.

1.6 Challenges with Natural Language (NL) Requirements and proposed solution

As mentioned, Natural Language (NL) has been used for capturing requirements as a means to make them more accessible to stakeholders [11]. This is in comparison to requirements defined in modeling languages such as Unified Modeling Language (UML) and Systems Modeling Language (SysML), which require special training [18]. While the benefits offered by NL are alluring, they present some challenges.

NL can be *ambiguous* [11], [19], [20] – capable of being understood in two or more ways [21]. For example – The display must have a *"user-friendly"* interface. Here, the word *"user-friendly"* can mean different things to different people, hence leading to ambiguities in requirements.

The second problem associated with NL requirements is *unrecognized disambiguation* [22] – the reader uses the first meaning that comes to their mind as the only meaning of a certain word, abbreviation, sentence, etc. For example: there is a difference in meaning between the two words *"FARs"* (Federal Aviation Regulations) and *"FARS"* (Fatality Analysis Reporting System). The reader would have completely misunderstood the context if they understood FARs as Fatality Analysis Reporting System (FARS) when talking about the aerospace domain and vice-versa when referring to self-driving car systems.

In addition, manual examination of NL requirements for checking completeness and consistency is tedious, and the effort goes up exponentially when the number of requirements increases [23]. Inconsistent, missing, and duplicate requirements are hard to fix manually [11] and can lead to catastrophic outcomes such as the failure of the Mars climate orbiter in 1998 due to confusion regarding units between the stakeholders involved in the project [24].

Lastly, failure to recognize technical terms and their meanings can lead to a reduced understanding of requirements for a particular domain. For example: not understanding and using terms effectively such as FAA, NASA, and ATC in the aerospace domain can lead to reduced requirements quality by introducing ambiguities and inconsistencies.

Most of the systems in the present-day world are complex and hence need a comprehensive approach to their design and development [25]. To accommodate this need, there has been a drive toward the development and use of Model-Based Systems Engineering (MBSE) principles and tools, where activities that support the system design

process are accomplished using models as compared to traditional document-based methods [26]. Models capture the requirements as well as the domain knowledge and make them accessible to all stakeholders [27], [28]. While MBSE shows great promise, the ambiguities and inconsistencies inherent to NL requirements hinder their direct conversion to models [29]. Hand-crafting models are time-consuming and require highly specialized subject matter expertise. As a result, there is a need to convert NL requirements into a semi-machine-readable form (which involves being able to extract information from NL requirements as well as converting them into a standardized form) so as to facilitate their integration and use in an MBSE environment. The need to access data within requirements rather than treating the statement as a standalone object has also been recognized by the International Council on System Engineering's (INCOSE) Requirements Working Group in their recent publication [30]. The document envisions an informational environment where requirements are not linked only to each other or test activities but also to architectural elements. This is represented in the figure below, which envisions a web of interconnections from a new kind of requirement object, which is a fusion of the natural language statement and machine-readable attributes that can connect to architectural entities such as interfaces and functions.

The approach of creating and maintaining requirements as more information-rich objects than natural language sentences has been called "Information-Based Needs and Requirements Definition and Management" (I-NRDM). In the INCOSE manual [30], model-based design (working on architecture and analysis) is recommended to combine with I-NRDM to be the full definition of MBSE. The "Property-Based Requirement" of recent SysML revisions can serve as the "Requirements Expression", as shown in Figure 1.5.

Despite these identified needs and ongoing developments, there are no standard tools or methods for converting NL requirements into a machine-readable/semi-machine-

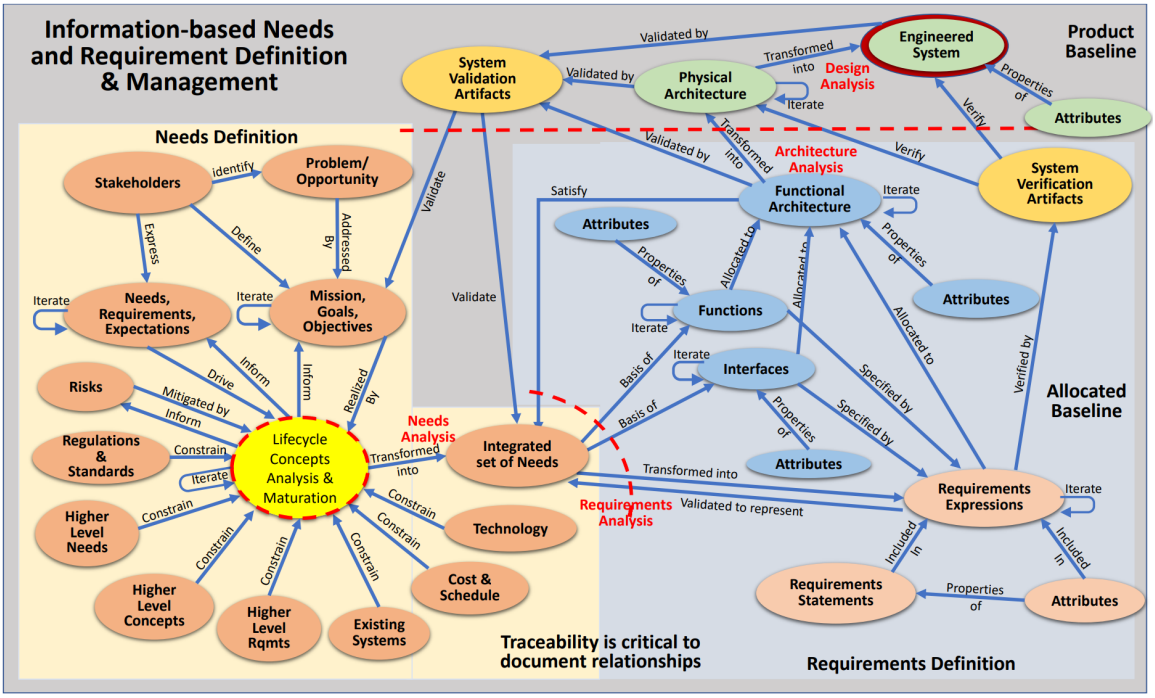


Figure 1.5: Information-based Requirement Development and Management Model [31]

readable form.

Observation 3

As systems become more complex and the number of requirements increases, it becomes difficult to evaluate requirement completeness and consistency manually, hence the need for automatic evaluation of requirements arises.

1.7 Summary

The requirements engineering phase of a project is very crucial and ambiguities in this phase can affect various downstream tasks such as system architecting, system design, testing, analysis, and inspection [14], ultimately resulting in a reduced quality system, cost overruns, and system failure [13]. The cost of fixing errors in requirements goes up exponentially as we move forward in the design and development process for a system [15].

Most of the requirements are written in natural language because of the low barrier to adoption as compared to model-based languages such as UML and SysML [11], [18]. Despite this advantage, NL requirements present a large number of drawbacks such as ambiguities associated with language [11], [19], [20], tedious and error-prone manual examination process, difficulties associated with verifying system description for completeness [23], and failure to recognize and use technical terms effectively that are associated with a domain [11].

To address the drawbacks associated with NL requirements there has been a shift towards machine-readable/semi-formal requirements, where data within requirements can be accessed as compared to treating requirement sentences as standalone objects [30].

This leads us to the research objective for this thesis:

Research Objective

Identify, develop, and implement tools and techniques to enable/support the automated translation of natural-language aerospace requirements into semi-machine-readable requirements.

For the purpose of this work, **semi-machine-readable** requirements means two things:

- Converting contents of NL requirements into data objects (Figure 1.6, Figure 1.7).
- Impart a structured format (predefined linguistic pattern) to freeform NL requirements. This keeps the requirements accessible to various stakeholders who might lack knowledge about model-based terminologies (Figure 1.7).

The air-taxi shall have a configuration that can seat five passengers within its passenger cabin.
SYSTEM = air-taxi, passenger cabin
VALUE = five passengers

Figure 1.6: An illustration of how to convert the contents of a natural language requirement into data objects can be seen in this example: In the given requirement "*The air-taxi shall have a configuration that can seat five passengers within its passenger cabin*", the *air-taxi* and *passenger cabin* are considered as SYSTEM, while *five passengers* is classified as a VALUE.

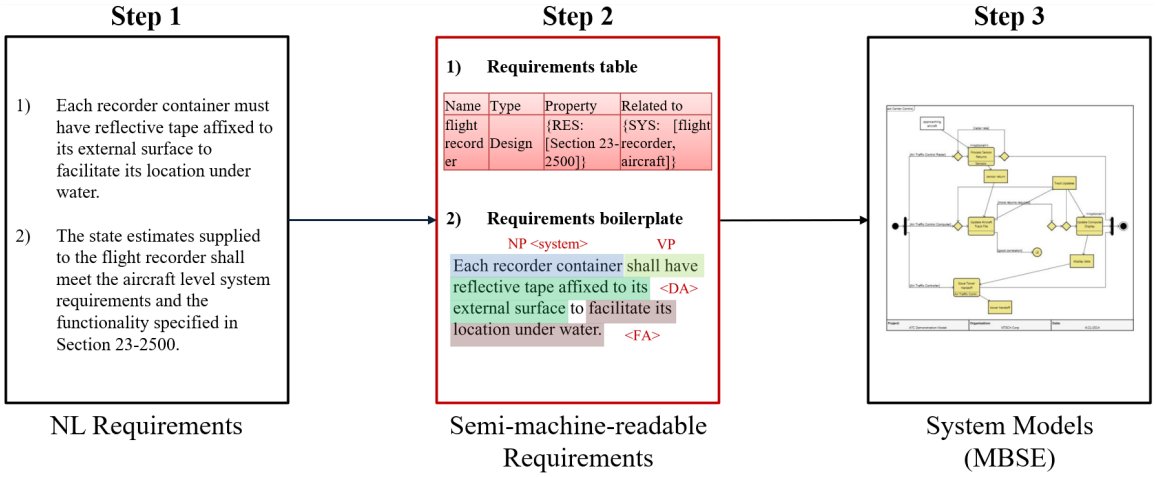


Figure 1.7: Different steps of requirements engineering, starting with gathering requirements from various stakeholders, followed by using Natural Language Processing (NLP) techniques to standardize them and lastly converting the standardized requirements into models. The main focus of the dissertation was to convert NL requirements into semi-machine-readable requirements (where parts of the requirement become data objects) as shown in Step 2.

1.8 Dissertation Overview

This dissertation focuses on developing a methodology for standardizing NL aerospace requirements by making use of transformer-based LMs. The anticipated outcome of this is to reduce the time and cost needed for requirements elicitation during the *Requirements and Functional Analysis* stage of the IPPD process, ultimately expediting the overall design and development process. The certification requirements from Parts 23 and 25 of Title 14 Code of Federal Regulations (CFRs) are the main use case to demonstrate the methodology which is generalizable. The dissertation is organized as follows:

- **Chapter 1** provides an introduction to the field of requirements engineering. It discusses the importance of NL in requirements elicitation and also its drawbacks. Lastly, this chapter sets the research objective for this dissertation, based on various observations.

- **Chapter 2** provides the background into the historical and current-day use of NLP in the field of requirements engineering. In addition, it discusses the relevance of pre-trained LMs and domain-specific corpus for analyzing aerospace requirements. The chapter concludes by outlining the observations made and identifying gaps in the current literature.

- **Chapter 3** outlines the research plan for this dissertation, which is based on the observations and identified gaps in the previous chapters. The chapter identifies the research questions and their corresponding hypotheses.

- **Chapter 4** describes the methodology for executing the experimental plan devised in the previous chapter. The methodology includes the creation of annotated aerospace corpora which is followed by fine-tuning BERT LM for NER, requirements classification, and POS tagging. Lastly, the chapter concludes by describing the methodology for using the models developed to standardize requirements.

- **Chapter 5** discusses the implementation of the proposed methodology in a detailed manner. This includes the creation of the corpora, converting texts in a format for fine-tuning BERT variants for different downstream tasks such as aerospace NER (aeroBERT-NER), classification of aerospace requirements (aeroBERT-Classifier), and text chunking. Lastly, requirements are converted into standardized form after using the LMs developed in the previous steps, namely, the creation of a requirements table, and identification of boilerplates.

- **Chapter 6** discusses the results regarding the various LMs that were developed for the conversion of NL requirements into standardized requirements. it also provides a comparison between the performance of these models to that of off-the-shelf models. Lastly, the requirements table and the identified boilerplates are presented and explained.

- **Chapter 7** offers a condensed version of the methodologies (using flowcharts) devised in this dissertation, with a focus on the industry audience, in order to facilitate the implementation of these methods in their respective domains.

- **Chapter 8** summarizes the findings and contributions of this dissertation, discusses its limitations, and suggests avenues for future work.

1.9 Publications

Excerpts from the following publications are included in this dissertation.

- Tikayat Ray, A., Pinon Fischer, O.J., Mavris, D.N., White, R.T. and Cole, B.F., "aeroBERT-NER: Named-Entity Recognition for Aerospace Requirements Engineering using BERT", AIAA 2023-2583. AIAA SCITECH 2023 Forum. January 2023.

- Tikayat Ray, A., Cole, B.F., Pinon Fischer, O.J., White, R.T., and Mavris, D.N., "aeroBERT-Classifier: Classification of Aerospace Requirements Using BERT", MDPI Aerospace 2023, 10, 279.

- Tikayat Ray, A., Pinon Fischer, O.J., White, R.T., Cole, B.F., and Mavris, D.N., "aeroBERT-NER: Aerospace Corpus and Language Model for Named-Entity-Recognition for Aerospace Requirements Engineering", [Under Review].

CHAPTER 2
PRELIMINARIES AND LITERATURE REVIEW

This chapter provides the necessary background into the historical and current-day use of NLP in the field of requirements engineering. The following sections and subsections describe the evolution of NLP and establish the relevance of pre-trained large language models (LLMs), and importance of domain-specific corpus for analyzing aerospace requirements. Various NLP tasks such as named entity recognition (NER), classification, etc. are discussed in regards to aerospace requirements. Lastly, this chapter summarizes the observations and identifies the gaps in literature.

2.1 Natural Language Processing for Requirements Engineering (NLP4RE)

Requirements are almost always written in NL [32] to make them accessible to different stakeholders. According to various surveys, NL was deemed to be the best way to express requirements [33], and 95% of 151 software companies surveyed revealed that they were using some form of NL to capture requirements [12]. Given the ease of using NL for requirements elicitation, researchers have been striving to come up with NLP tools for requirements processing dating back to the 1970s. Tools such as the Structured Analysis Design Technique (SADT), and the System Design Methodology (SDM) developed at MIT, are systems that were created to aid in requirement writing and management [33]. Despite the interest in applying NLP techniques and models to the requirements engineering domain, the slow development of natural language technologies thwarted progress until recently [11]. The availability of NL libraries/toolkits (Stanford CoreNLP [34], NLTK [35], spaCy [36], etc.), and off-the-shelf transformer-based [37] pre-trained language models (LMs) (BERT [38], BART [39], etc.) have propelled NLP4RE into an active area of research [32]. NLP4RE deals with applying

Natural Language Processing (NLP) tools and techniques to the field of requirements engineering [40].

A recent survey performed by Zhao et al. reviewed 404 NLP4RE studies conducted between 1983 and April 2019 and reported on the developments in this domain [40]. Figure 2.1 shows a clear increase in the number of published studies in NLP4RE over the years. This underlines the critical role that NLP plays in requirements engineering, a role that is expected to become more important with time as the availability of off-the-shelf language models increases.

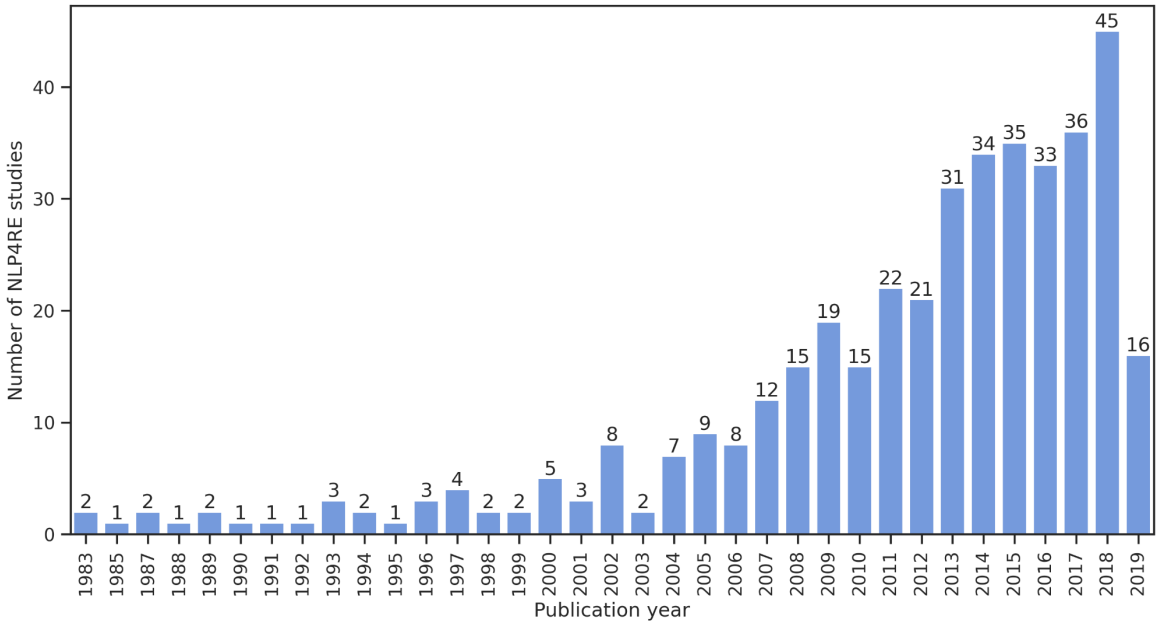


Figure 2.1: Timeline of published studies in NLP4RE [40]

Among those 404 NLP4RE studies, 370 NLP4RE were classified based on the main NLP4RE task being performed. As illustrated in Figure 2.2, the most common focus of the studies was the detection of linguistic issues in requirements, such as the occurrence of ambiguous phrases, the conformance to pre-defined templates, etc. The classification task was the second most popular task and dealt with the classification of requirements into various categories. Extraction dealt with the identification of key domain concepts. Finally, modeling, tracing and relating, and search and retrieval are some of the other NLP4RE tasks of interest.

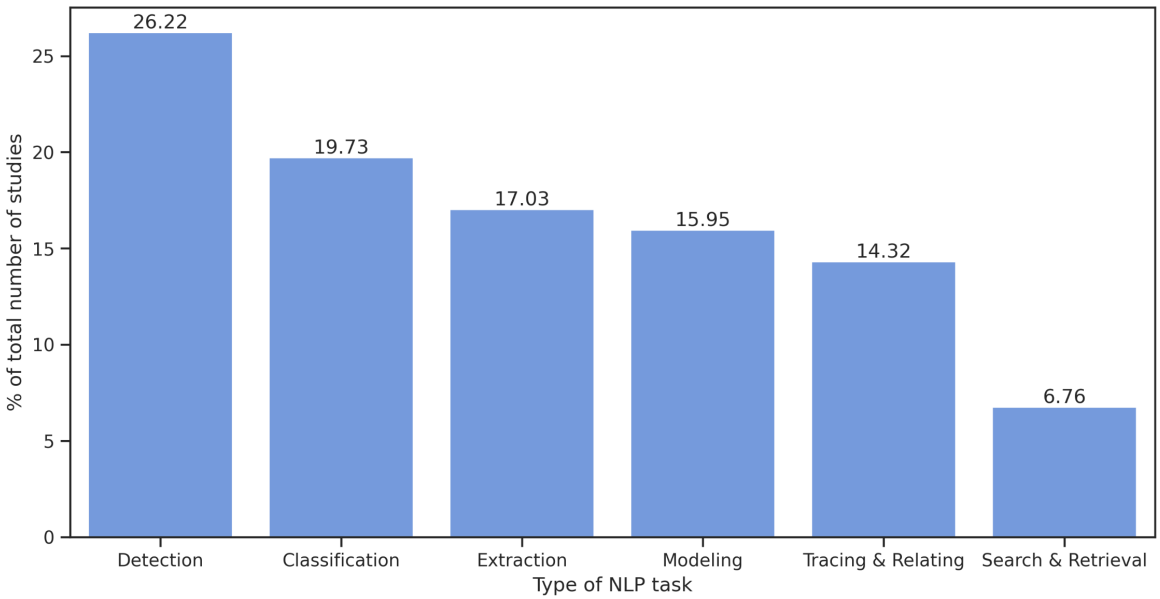


Figure 2.2: Distribution of selected studies based on NLP4RE task [40]

Despite the growing popularity of NLP4RE as a research domain, some gaps remain, as listed below [40]:

- **Lack of industrial case studies:** A divide prevails between NLP4RE research and its industrial penetration [11], [41]–[44]. This divide can be attributed to the fragmented approach to sharing knowledge pertaining to NLP4RE tools, techniques [45], and datasets.

- **Lack of open-source requirements datasets:** The shortage of open-source annotated datasets in requirements engineering, especially in the context of aerospace applications, has impeded the utilization of contemporary language models for tasks such as requirements classification and identifying domain-specific named entities. Additionally, since annotating requirements datasets necessitates subject-matter expertise, crowd-sourcing is not a feasible approach for data collection. In a study by Gilardi et al. [46], zero-shot ChatGPT (a large language model) was used to annotate tweets, which contain general-purpose text that ChatGPT was trained on. The results showed that ChatGPT

outperformed the crowd-workers on Amazon Mechanical Turk (MTurk) in four of the five tasks. However, the success of ChatGPT for text annotation is not expected to extend to technical domains.

- **Lack of the use of advanced NLP techniques for analysis of aerospace requirements:** Advanced LMs such as BERT, GPT-2 [47], etc. have been used to analyze software requirements [48], [49], however, no such effort has been carried out in the aerospace requirements engineering domain.

Advanced NLP tools and techniques have the potential to revolutionize NLP4RE research (a field that is known to have limited annotated datasets) in the aerospace requirements engineering domain and are the subject of this dissertation.

2.2 NLP and Case for Aerospace specific corpus

Language processing can aid in the conversion of NL requirements into semi-machine-readable requirements which will support the paradigm shift towards MBSE, where activities that support the system design process are accomplished using detailed models as compared to traditional document-based methods [26]. This section provides a brief overview of concepts such as NLP, Language Models (LMs), corpus, and the importance of domain-specific corpus.

2.2.1 Natural Language Processing (NLP)

Definition: "*Natural Language Processing (NLP)* is a theoretically motivated range of computational techniques for analyzing and representing naturally occurring texts at one or more levels of linguistic analysis for the purpose of achieving human-like language processing for a range of tasks or applications [50]."

NLP lies at the intersection of computer science and linguistics. Examples of some NLP tasks are speech recognition, word sense disambiguation, sentiment analysis,

spell check, etc.

NLP can be further divided into *Natural Language Understanding (NLU)* and *Natural Language Generation (NLG)* [51] (Figure 2.3). NLU involves understanding and interpreting text whereas NLG uses the meaning of text (as understood by NLU) in order to produce text [51].

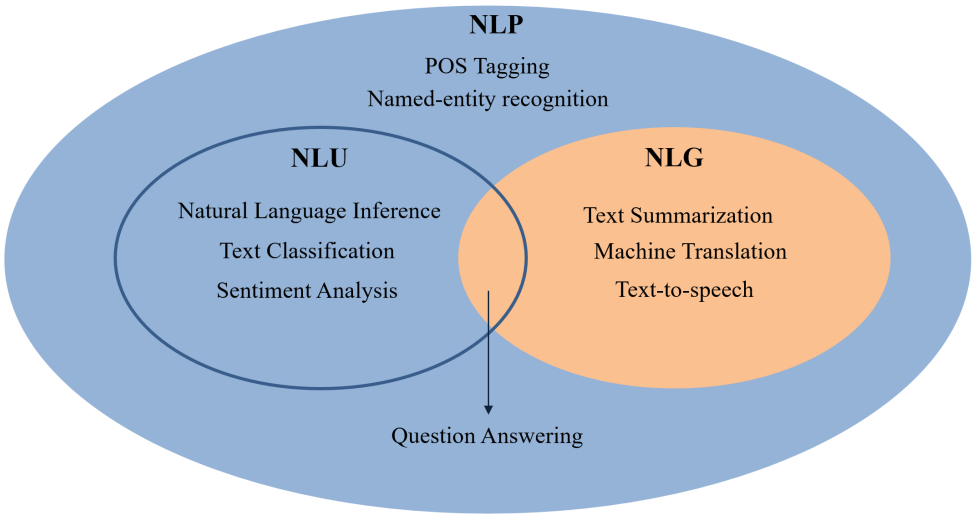


Figure 2.3: NLP and its subfields [51]

NLP tools and techniques have a strong potential to enable/support the automatic translation of NL requirements into machine-readable requirements by extracting and converting information present in a requirement into data objects. For example, word sense disambiguation and Named-entity recognition (NER) can help with reducing ambiguities associated with requirements; parts-of-speech (POS) tagging, text chunking, and dependency parsing can aid the requirements examination or re-write process; classification of requirements and boilerplate identification will be crucial for standardizing requirements (Figure 2.4). These aspects are discussed in the following sections.

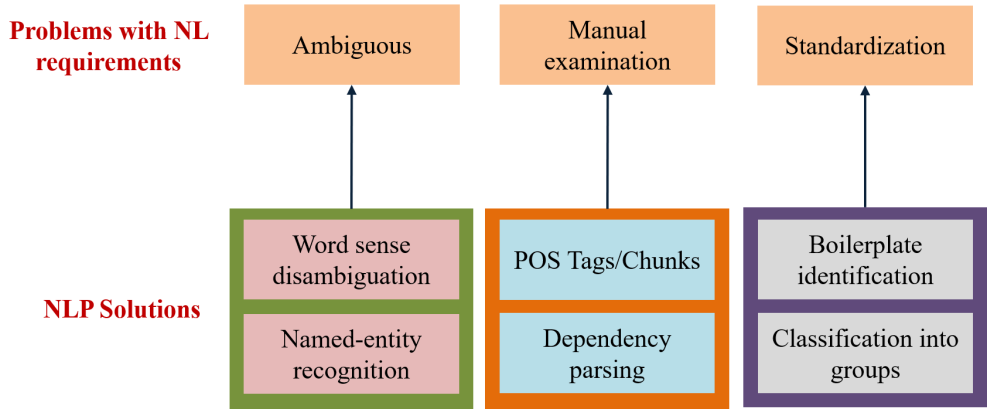


Figure 2.4: Problems with NL requirements and potential solutions provided by NLP

2.2.2 Language Model (LM) and corpus

NLP is promising when it comes to requirements analysis, which is a potential step toward the development of pipelines that can convert free-form NL requirements into standardized requirements in a semi-automated manner. Language models (LMs), in particular, can be leveraged for classifying text, extracting named entities (NEs) of interest [38], [39], [52], etc. Language modeling was classically defined as the task of predicting which word comes next [53]. Initially, this was limited to statistical language models [54], which use prior word sequences to compute the conditional probability for each of a vocabulary future word. The high-dimensional discrete language representations limit these models to N-grams [55], where only the prior N words are considered for predicting the next word or short sequences of following words, typically using high-dimensional one-hot encodings for the words (Figure 2.5).

Neural LMs came into existence in 2000s [56] and leveraged neural networks to simultaneously learn lower-dimensional word embeddings and learn to estimate conditional probabilities of next words simultaneously using gradient-based supervised learning. This opened the door to ever-more-complex and effective language models to perform an expanding array of NLP tasks, starting with distinct word embeddings [57] to recurrent neural networks (RNNs) [58] and LSTM encoder-decoders [59] to

attention mechanisms [60]. These models did not stray too far from the N-gram statistical language modeling paradigm, with advances that allowed text generation beyond a single next word with for example beam search in [59] and sequence-to-sequence learning in [61]. These ideas were applied to distinct NLP tasks.

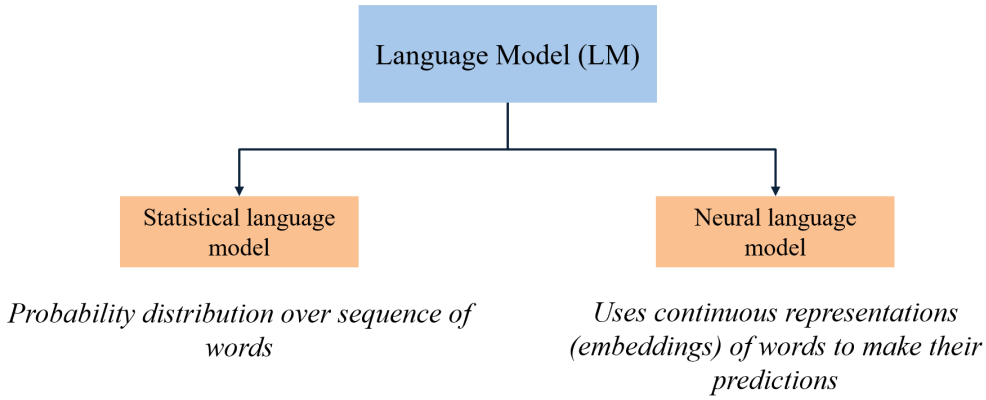


Figure 2.5: Language Model and its types

Neural language models have proven to be more effective as compared to statistical language models [62] and hence the rest of the document will focus on them. More details about statistical language models can be found in [54].

In 2017, the Transformer [37] architecture was introduced that improved computational parallelization capabilities over recurrent models, and therefore enabled the successful optimization of larger models. Transformers consist of stacks of encoders (encoder block) and stacks of decoders (decoder block), where the encoder block receives the input from the user and outputs a matrix representation of the input text. The decoder takes the input representation produced by the encoder stack and generates outputs iteratively [37].

All of these works required training on a single labeled dataset for a specific task. In 2018-20, several new models emerged and set new state-of-the-art marks in nearly all NLP tasks. These transformer-based models include the Bidirectional Encoder Representational Transformer (BERT) language model [38] (auto-encoding model), the Generative Pre-trained Transformer (GPT) family [47], [63] of auto-regressive

language models, and T5 character-level language model [64]. These sophisticated language models break the single dataset-single task modeling paradigm of most mainstream models in the past. They employ self-supervised pre-training on massive *unlabeled* text corpora. For example, BERT is trained on Book Corpus (800M words) and English Wikipedia (2500M words) [38]. Similarly, GPT-3 is trained on 500B words gathered from datasets of books and the internet [63].

These techniques set up automated supervised learning tasks, such as masked language modeling (MLM), next-sentence prediction (NSP), and generative pre-training. No labeling is required as the labels are automatically extracted from the text and hidden from the model, and the model is trained to predict them. This enables the models to develop a deep understanding of language, independent of the NLP task. These pre-trained models are then fine-tuned on much smaller labeled datasets, leading to advances in the state-of-the-art (SOTA) for nearly all downstream NLP tasks (Figure 2.6), such as Named-entity recognition (NER), text classification, language translation, question answering, etc. [38], [65], [66].

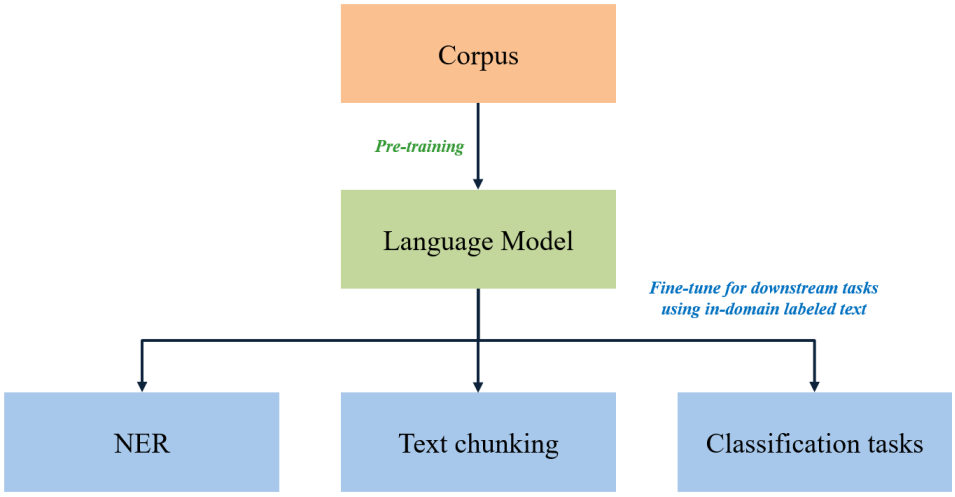


Figure 2.6: Chronology of Corpus, Language Model, and downstream tasks

2.2.3 Importance of domain-specific corpus

As previously mentioned, LMs are often pre-trained on a general-domain corpus (news articles, Wikipedia, books). As a result, typical off-the-shelf LMs that have been fine-tuned for downstream tasks on general-domain corpora are not effective when applied to technical domains, such as aerospace, biomedical engineering, etc. Their application leads to poor and unacceptable results [67], [68], such as failure to recognize important in-domain terms such as FAA, FARs, AoA, etc. [69], which in turn impede the usefulness of such LMs when used for aerospace requirements analysis. In addition, most datasets related to sentiment analysis, movie reviews, etc., are either automatically labeled by extracting star ratings on movie/product reviews or crowd-annotated since they contain only general-purpose language. However, pre-annotated datasets rarely exist for aerospace engineering (specifically, requirements engineering). Further, annotating text from these domains requires subject matter expertise. Hence, there is a lack of publicly annotated datasets.

There are two approaches to solving this problem with modern LMs. The first approach is to use an LM pre-trained on a general-domain unlabeled text corpus and fine-tune it for specific supervised downstream tasks like NER with a labeled in-domain corpus [65]. The second approach is to pre-train the LM from scratch using an unlabeled in-domain text corpus and fine-tune it to the downstream task. This approach is helpful when dealing with domains that are substantively different from a general-domain text corpus, hence hindering transfer learning performance [68]. The first approach requires far less effort and resources and is preferred when it proves effective.

Observation 4

Language models are trained on general-domain text, which leads to poor performance by these models when applied to aerospace requirements due to a lack of domain knowledge.

2.3 Identification of named entities (NEs) in aerospace requirements

Often stakeholders use different terms to refer to the same entity, hence leading to ambiguities [11], [19], [20]. As such there is a need to build a glossary for terms occurring in aerospace requirements, however, doing this manually is an arduous task when dealing with large number of requirements [69]. According to Arora et al. [69], a glossary can be obtained by putting NL requirements through the NLP pipeline, which first tokenizes the text, followed by POS tagging, NER and chunker as shown in Figure 2.7. The output produced by the pipeline are annotated tokens (POS, NE, and text chunks) out of which only Noun-Phrases (NPs) are of interest. *Noun Phrases (NPs)* can be object or subject of a verb. *Verb Phrases (VPs)* consists of the verb along with modal, auxiliary, and modifier.

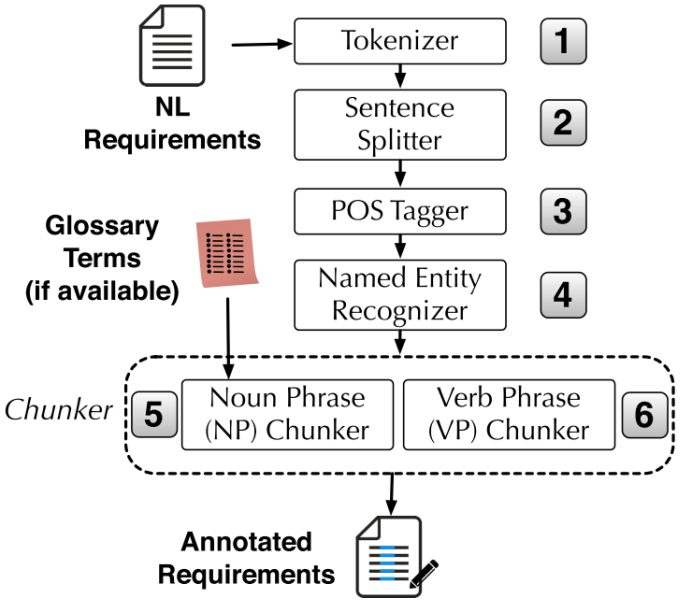


Figure 2.7: NLP pipeline for text chunking and NER [69]

Arora et al. [69] carried out a clustering task to account for different variations of a term. For example, "status of the air-taxi" and "Air-taxi status" mean the same even though the chronology of tokens/phrases is different. This work focuses only on extracting and including NPs in the glossary which is one of its limitations. Second limitation, the authors used different tools in conjunction (openNLP, GATE, and JAPE heuristics for term extraction; SimPack, SEMILAR for similarity computation; and R for selecting the number of clusters) to facilitate the creation of clusters of extracted terms as compared to providing a list of glossary terms.

Hence, to mitigate the vagueness and ambiguities of aerospace requirements, a methodology is needed for a more comprehensive glossary creation that contains different types of NEs (such as names of organizations, systems, resources, values, etc.) pertaining to the aerospace domain. Due to the advancements in NLP, the feasibility of a comprehensive and automated tool for glossary creation also needs to be explored.

Observation 5

Stakeholders use different terms/words to refer to the same entity/idea when framing requirements leading to ambiguities.

The ultimate goal is to be able to convert aerospace NL requirements into machine-readable requirements. A stepping stone to this is to be able to classify requirements into various types (functional, non-functional, data requirements, etc.). Functional requirements (FRs) describe the functionality of a system whereas non-functional requirements (NFRs) describe the properties and constraints associated with a system [70]–[72]. Being able to classify requirements can help with focused communication between different stakeholders, prioritization of requirements, filtering requirements based on type, and aiding in the verification and validation process [73]–[75]. Oftentimes, requirements are reused for different design components and systems, and

hence being able to classify them makes this task easier [76]. The following section will discuss more about requirements classification.

2.4 Classification of aerospace requirements

Classification of software requirements has been extensively studied. Cleland-Huang et al. [77] describe the importance of non-functional requirements (NFRs) and developed a method for extracting them from requirement specification documents, as well as from free-form documentation such as meeting minutes, memos containing stakeholder needs, etc. Their methodology consists of three steps: mining, classification, and application. During the mining phase, terms contained in NFRs are mined, which are then used to classify requirements during the classification phase. The classified requirements support activities such as architectural design and requirements negotiation in the application phase [77]. Despite missing out on classifying all the NFRs, this method is helpful as compared to manual analysis of requirements [78]. Rashwan et al. [79] created a corpus consisting of different classes of annotated NFRs (constraint, usability/utility, security, efficiency, functionality, reliability, maintainability) and went on to develop a *Support Vector Machine* (SVM) classifier for classifying these NFRs.

Winkler and Vogelsang [80] used Convolutional Neural Networks for the classification of elements of a requirement specification into either *requirement* or *information*. The requirement class consisted of parts of requirement specifications that could be verified, whereas the information class contained explanations, summaries, references to other documentation, and examples.

Classification of requirements has been proven to be important in case of security requirements such as analyzing Software Requirement Specification (SRS) documents. Jindal et al. [81] proposed a methodology for classifying security requirements into four classes such as authentication-authorization, access control, cryptography-

encryption, and data integrity by using J48 decision trees. They removed stopwords from the security requirement descriptions, which might not be a good idea when it comes to analyzing aerospace requirements. Indeed, terms like *not*, *must*, *when*, etc. are stop-words and add meaning to requirement specifications.

The field of requirements classification has moved from manual to semi-automatic to automatic by making use of breakthroughs in the field of machine learning (ML). A literature review published by Binkhonain et al. [82] looked into 24 selected studies, all of which used ML algorithms for analyzing NFRs. Out of these 24 studies, 17 used supervised learning (SL) (71%) and SVM proved to be the most popular ML algorithm to be used. All the studies used pipelines, where the first step was to pre-process NL requirements, followed by a feature extraction phase, a learning phase where the ML models were trained, and the last step was model evaluation, where the model's performance was evaluated on a test dataset [82]. The pipeline is represented in Figure 2.8.

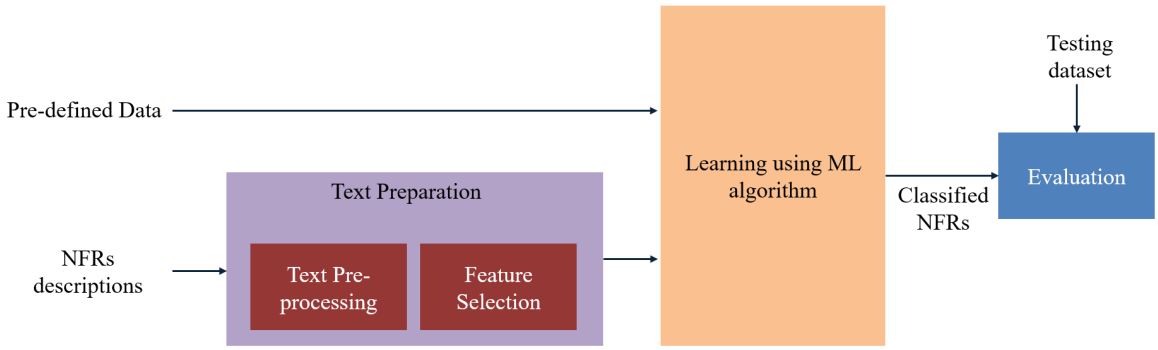


Figure 2.8: Pipeline for classifying NFRs using ML [82]

In a recent study, Hey et al. fine-tuned the BERT language model on the PROMISE NFR dataset [49] to obtain NoRBERT (Non-functional and functional Requirements classification using BERT) - a model capable of classifying requirements [83]. NoR-BERT is capable of performing four tasks, namely, (1) binary classification of requirements into two classes (functional and non-functional); (2) binary and multi-class classification of four non-functional requirement classes (Usability, Security,

Operational, and Performance); (3) multi-class classification of ten non-functional requirement types; and (4) binary classification of the functional and quality aspect of requirements. NoRBERT was able to achieve an average F1 score of 0.87 on the most frequently occurring classes of requirements (Usability, Performance, Operational, and Security). In particular, it demonstrates the relevance and potential of transfer-learning approaches to requirements engineering research as a means to address the limited availability of labeled data. The PROMISE NFR dataset, [49], which was used in [83] contains 625 requirements in total (255 functional and 370 non-functional, which are further broken down into different "sub-types"). Table 2.1 provides some examples from the PROMISE NFR dataset.

Table 2.1: Requirements examples from the PROMISE NFR dataset [49]

Serial No.	Requirements
1	The product shall be available for use 24 hours per day 365 days per year.
2	The product shall synchronize with the office system every hour.
3	System shall let existing customers log into the website with their email address and password in under 5 seconds.
4	The product should be able to be used by 90% of novice users on the Internet.
5	The ratings shall be from a scale of 1-10.

These requirements originated from 15 projects written by students and as a result, might not have been written according to industry standards. The PROMISE NFR dataset is, to the authors' knowledge, the only requirements dataset of its kind that is publicly available. However, this dataset was not deemed suitable for this work because it predominantly focuses on software engineering systems and requirements.

Another avenue for text classification is Zero-Shot-Learning (ZSL), where models performing a task have not been explicitly trained for the task [84]. Such models, due to their nature, provide a logical basis for the evaluation of LMs that have been previously fine-tuned on requirements specific to a discipline of interest.

There are two general ways for ZSL, namely, *entailment-based* and *embedding-*

based methods. Yin et al. proposed a method for zero-shot text classification using pre-trained Natural Language Inference (NLI) models [85]. The *bart-large-mnli* model was obtained by training *bart-large* [39] on the MultiNLI (MNLI) dataset, which is a crowd-sourced dataset containing 433,000 sentence pairs annotated with textual entailment information [0: entailment; 1: neutral; 2: contradiction] [86]. For example, to classify the sentence "*The United States is in North America*" into one of the possible classes, namely, *politics*, *geography*, or *film*, we could construct a hypothesis such as - *This text is about geography.* The probabilities for the entailment and contraction of the hypothesis will then be converted to probabilities associated with each of the labels provided.

Alhosan et al. [48] performed a preliminary study for the classification of requirements using ZSL in which they classified non-functional requirements into two categories, namely usability, and security. An embedding-based method was used where the probability of the relatedness (cosine similarity) between the text embedding layer and the tag (or label) embedding layer was calculated to classify the requirement into either of the two categories. This work which leveraged a subset of the PROMISE NFR dataset was able to achieve a recall and F-score of 82%. The authors acknowledge that certain LMs ($\text{RoBERTa}_{\text{Base}}$ and XLM-RoBERTa) seemed to favor one or the other class, hence classifying all the requirements as either usability or security. This was attributed to the fact that LLMs are trained on general domain data and hence might not perform well on specialized domains [49].

The classification of requirements is an important task and there exists extensive research when it comes to software requirements. However, research pertaining to leveraging requirements classification for consistency and redundancy checks, and identification of requirements boilerplates is scarce, both in software and aerospace domains. Hence, ideas from the above literature can be leveraged to classify aerospace requirements which will be a stepping stone toward converting aerospace NL require-

ments into semi-machine-readable requirements.

Observation 6

There is limited work on the classification of aerospace requirements which hinders the ability to conduct redundancy checks, evaluate consistency, and standardization of requirements.

2.5 Standardization of aerospace requirements and use of boilerplates

As mentioned previously, requirements are articulated in NL because of the universality it offers. Several researchers have proposed the use of templates (mold or patterns) for improving the quality of requirements, reducing ambiguity and inconsistencies [18], [87]. *Requirement boilerplates* are pre-defined linguistic patterns [88] that can be applied to NL aerospace requirements to standardize them and eventually allow them be used for automated analysis. The positives of boilerplates are that the requirements are still in NL and hence are accessible to all stakeholders while having a well-defined syntactic structure [18] that facilitates their understanding - which takes us closer to machine-readable requirements.

As such, requirement boilerplates offer benefits in industrial settings and are considered of paramount importance when it comes to safety-critical systems [87]. Several projects concerned with the development and certification of safety-critical systems (CESAR [89], OPENCOSS [90], SAREMAN [87]) underline the importance of using boilerplates for requirements specification. Requirement authoring and management tools offer support for boilerplates [91], which also underscores their importance [87].

Two of the most popular boilerplates for standardizing requirements are Rupp's (Figure 2.9) and Easy Approach to Requirements Syntax (EARS) (Figure 2.10).

Rupp's boilerplate structure specifically focuses on functional requirements. The boilerplate structure accounts for three different types of system activities, namely,

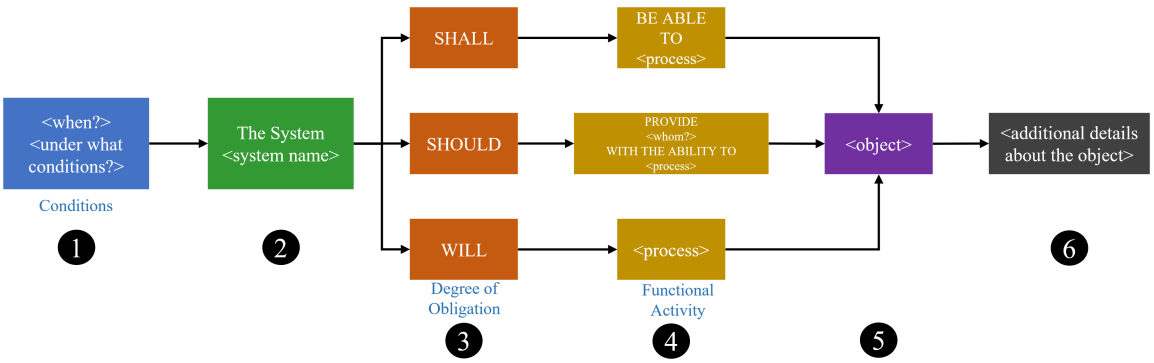


Figure 2.9: Rupp's Boilerplate [18], [88], [92]

interface, user interaction, and autonomous [93]. The different parts of this boilerplate structure are described below [18], [88]:

- **Condition**: can include single or multiple conditions (optional) - usually starting with *"if"*, *"as soon as"*, *"after that"*
- **System**: the name of the system or subsystem
- **Degree of obligation**: degree of obligation for the requirement (*"shall"* for mandatory requirements; *"should"* for recommended requirements; *"will"* for future requirements; *"may"* for desirable requirements)
- **Functional activity**: type of functionality that the systems intends to achieve which may include:
  1. performing a functionality independently
  2. providing some functionality to users
  3. functionality as a reaction to events with other systems
- **Object**: object for which behavior described in the requirement is executed
- **Additional details about the object**: more information regarding the object

Similarly, the EARS boilerplate can be divided into four parts [88]. The first part is an optional condition block, followed by the system/subsystem name, the degree of obligation, and the response of the system.

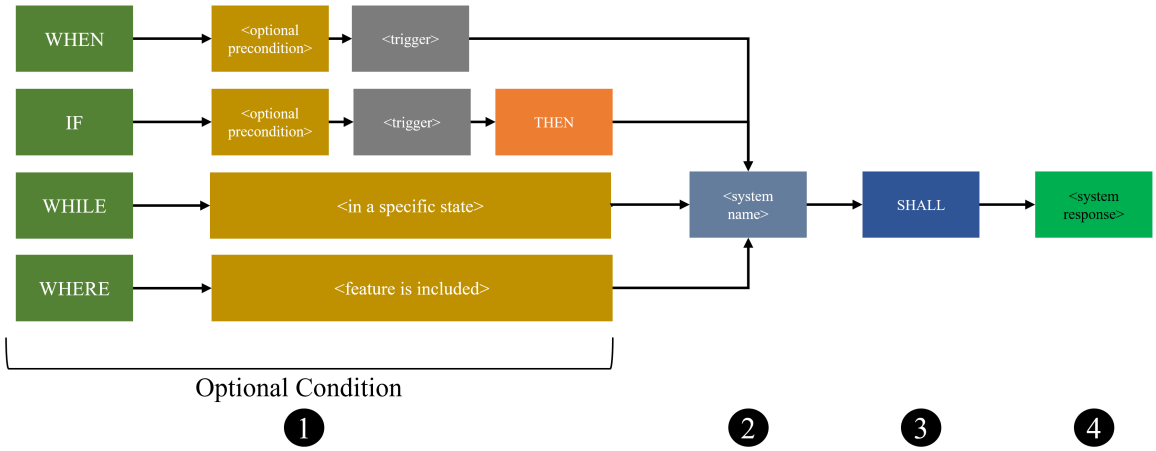


Figure 2.10: EARS Boilerplate [88], [94]

Despite the pros offered by requirement boilerplates (Rupp's and EARS), they can be restrictive when it comes to certain types of functional and non-functional requirements, and do not allow for the inclusion of constraints [18]. In some cases, using Rupp's boilerplate can lead to inconsistencies due to the restrictions imposed by the structure leading to exclusion of ranges of values and bi-conditionals, lack of reference to external systems that the original system interacts with [18]. Mazo et al. [18] improved on Rupp's template and came up with a new boilerplate that addresses the shortcomings of Rupp's. It consists of eight blocks as compared to six in Rupp's [Figure 2.11].

Observation 7

Research focusing on the definition of appropriate, relevant, and reliable boilerplate is scarce; Rupp's and EARS boilerplates do not capture all aspects of a requirement.

Arora et al. [95] stress the importance of verifying whether requirements actually conform to a given boilerplate, which is important for quality assurance purposes. In

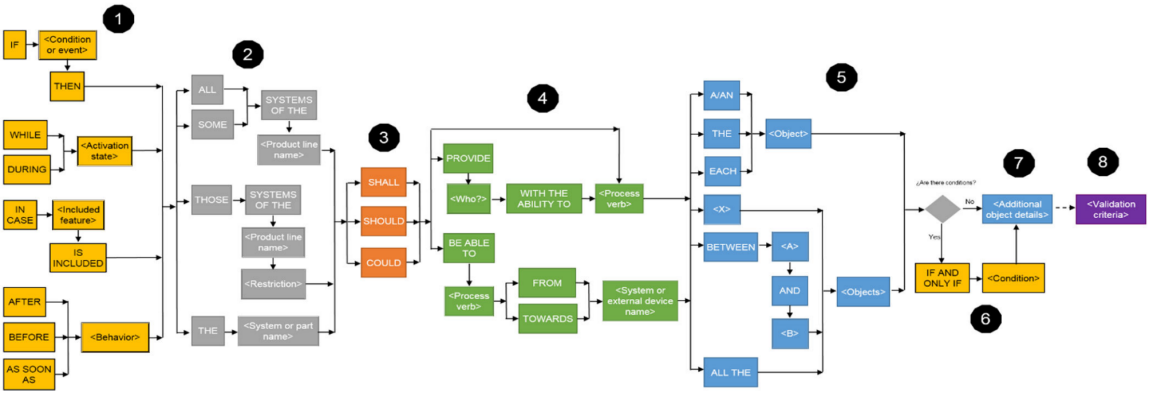


Figure 2.11: Majo and Jaramillo template [18]

their paper, they check the conformance of requirements to Rupp's boilerplate using *text chunking* (NPs and VPs) [95]. In addition, they suggest that this is a robust method for checking conformance in the absence of a glossary [95]. In another paper by the same authors [88], they develop a conformation check methdology for Rupp's and EARS boilerplates using the NLP pipeline described in Figure 2.7.

Boilerplate identification has been significantly limited due to variations in requirements across different industries and institutions. Furthermore, previous studies have primarily concentrated on software requirements, which can differ substantially from aerospace requirements, the focus of this study.

Observation 8

There has been limited work focusing on identifying appropriate boilerplates given a type of requirement e.g., converting a NL requirement into an appropriate boilerplate structure.

This study defines standardization of requirements as the capability to convert unstructured natural language requirements into a structured format that adheres to a predetermined sequence of elements identified through linguistic pattern analysis. For example, the format might generally commence with the system name and is accompanied by supplementary elements that provide additional context for the

requirement.

2.6 Summary of Observations and Gaps

Requirements are predominantly written in NL due to the flexibility that NL provides. However, NL also adds ambiguities and inconsistencies. Manually examining a large number of requirements is prohibitive. As such, there is a need for automating the process of standardization of requirements, which will make further analysis of requirements much easier and more efficient. In addition, standardized requirements are expected to facilitate their integration and use in MBSE environments. Research studies in this field do exist, however, they are scarce and focus mostly on software requirements. Hence, there is a need to further the research in this direction and focus on different domains (aerospace in our case).

The literature review leads to the following observations:

1. Requirements are written in Natural Language (NL) to make them more accessible to different stakeholders; however, this introduces unintended inconsistencies and ambiguities.

2. The cost of fixing errors in requirements goes up exponentially as we progress across the project life cycle.

3. As systems become more complex and the number of requirements increases, it becomes difficult to evaluate requirement completeness and consistency manually, hence the need for automatic evaluation of requirements arises.

4. Language models are trained on general-domain text, which leads to poor performance by these models when applied to aerospace requirements due to a lack of domain knowledge.

5. Stakeholders use different terms/words to refer to the same entity/idea when framing requirements leading to ambiguities.

6. There is limited work on the classification of aerospace requirements which hinders the ability to conduct redundancy checks, evaluate consistency, and standardization of requirements.

7. Research focusing on the definition of appropriate, relevant, and reliable boilerplate is scarce; Rupp's and EARS boilerplates do not capture all aspects of a requirement.

8. There has been limited work focusing on identifying appropriate boilerplates given a type of requirement e.g., converting a NL requirement into an appropriate boilerplate structure.

The above observations can be summarized into the following two gaps:

1. **Gap 1:** Lack of language models with aerospace-specific domain knowledge and need for an automated way to extract terms for the creation of glossary.

2. **Gap 2:** Lack of requirement analysis methods for automated classification of aerospace requirements and conversion of NL requirements into appropriate boilerplate structures.

CHAPTER 3
RESEARCH FORMULATION

This chapter describes the research plan for the dissertation based on observations, and gaps recognized in the last two chapters. The goal of this chapter is four folds, namely:

1. Identification of the research areas given the research objective, observations, and gaps from the literature review

2. Formulation of research questions and sub-questions based on the identified research areas to fill the gaps identified from literature

3. Formulation of hypotheses corresponding to the research questions which will be tested using experiments

4. Formulation of overarching hypothesis based on the research questions and their corresponding hypotheses

3.1 Research Objective

Chapter 1 describes the importance of the requirements engineering phase in a project life-cycle and the costs associated with bad requirements. Most requirements are written in NL because of the low barrier to adoption. However, despite this advantage, NL requirements can be ambiguous, and not standardized, making it hard to employ automated verification methods. This leads us to the research objective which is reiterated below.

Research Objective

Identify, develop, and implement tools and techniques to enable/support the automated translation of natural-language requirements into semi-machine-readable requirements.

Semi-machine-readable requirements are central to our ability to move from a document-centric to a model-centric approach to engineering. The literature review carried out in Chapter 2 suggests that converting NL requirements into a semi-machine-readable (or standardized) form requires the development and implementation of NLP techniques to facilitate the conversion.

Given the observations from the literature review, three sub-objectives were formulated and are listed below:

1. Fine-tune LMs with annotated aerospace-specific corpus to enable them to identify named entities (NEs), and tag parts-of-speech (POS tagging) specific to the aerospace domain

2. Fine-tune LM with annotated aerospace requirements corpus to enable the LM for classifying requirements

3. Use the LMs created in the previous step for standardization of aerospace requirements. This involves being able to extract information from requirements and converting them to data objects, as well as, identifying text patterns in NL requirements.

In order to achieve the above objectives, the gaps identified in Chapter 2 must be addressed. Hence, two research questions were formulated along with their respective sub-questions and corresponding hypotheses to achieve the objectives. A road map for this dissertation is outlined in Figure 3.1.

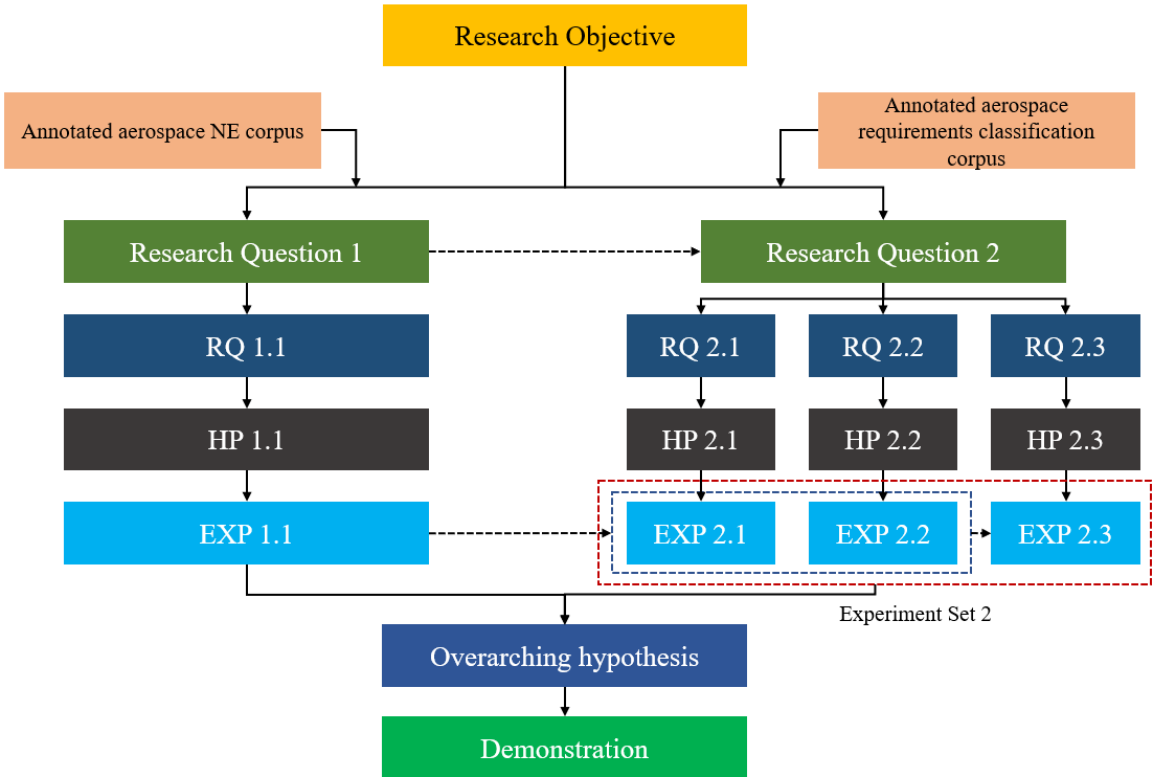


Figure 3.1: Research road-map

3.2 Research Question 1: Incorporating aerospace domain knowledge into LMs

Gap #1 - lack of LMs with aerospace-specific domain knowledge will be addressed by RQ 1. As described in Observations #4 and #5, LMs are trained on general-domain corpora, which leads to bad performance when these models are used in scientific/engineering domains such as aerospace. In addition, a case was made for the fact that stakeholders use different terms to refer to the same entity, hence leading to ambiguities in NL requirements. The above challenges lead to the following research question.

Research Question 1

How can engineering/aerospace-specific domain knowledge be integrated into language models?

To answer the research question posed above, corpus, LMs (BERT specifically), and fine-tuning of LMs need to be further discussed.

A *corpus* is a collection of texts, for example, all the text available on Wikipedia on which LMs can be trained.

As mentioned previously, a suitable corpus is required to train LMs [96]. However, there is a lack of annotated corpus when it comes to engineering domains [97], [98] such as aerospace making it difficult to tackle NLP tasks such as identification of NEs, classification of requirements, etc.

SciBERT is a well known language model trained on scientific text [97]. This language model was trained on a corpus that contains 18% of the text from the computer science domain and 82% from the biomedical domain [97]. The vocabulary size is 30,000 tokens which is the same as BERT and there is a 42% overlap between both vocabularies, meaning 58% of the words specific to the scientific domain were absent from the general-domain corpus [97].

FinBERT (Financial BERT) [99] uses a financial services corpus for training the LM; *BioBERT* [100] uses biomedical literature for training; *ClinicalBERT* [101] uses clinical notes; and *PatentBERT* [102] fine-tunes pre-trained BERT model for patent classification. The existence of these LMs stresses the fact that domain-specific corpus is crucial when it comes to domains that are not well represented in the general-domain text (newspaper articles, Wikipedia, etc.).

Hence, in the context of this research, a LM is needed that has the capability to perform well on aerospace text. Doing so involves the following two sub-tasks, namely:

1. Identifying a LM to work with
2. Creating an aerospace corpus

Training sophisticated LMs from scratch is difficult due to the computational

power necessary to do so. For example, GPT-3 required 355 GPU years and cost $4.6 million [67] to train. BERT has two pre-trained models - $\text{BERT}_{\text{BASE}}$ with 110 million parameters, which took between $2.5k - $50k to train and $\text{BERT}_{\text{LARGE}}$, which has 340 million parameters and took between $10k - $200k to train [103].

The high cost of training LMs makes it prohibitive to train a model from scratch for the purpose of this research, the focus will be on *fine-tuning* a LM for various NLP tasks using small labeled corpora. The task at hand is to choose a LM that satisfies certain criteria of interest, as shown in Table 3.1.

Table 3.1: Criteria considered when choosing a LM

Criteria	Description
Pre-trained neural LM	Ability to learn common language representations by using large corpus of unlabeled data in an unsupervised/semi-supervised manner [65]
Can be fine-tuned for a variety of tasks	Fine tuning is the process of tuning the pre-trained model's parameters by using data of interest – aerospace in our case
Bidirectional transformer-based architecture	Bidirectional architecture makes sure that the text is looked at in both the forward and backward direction in order to better capture the context; transformers make use of an attention mechanism to make the model training faster [104]
State-of-the-art (SOTA) results for NL tasks such as NER, text classification, etc.	State-of-the-art (SOTA) models can achieve higher scores when it comes to certain NLP tasks of interest

Considering the above criteria, Google's BERT LM was selected. BERT stands for ***B**idirectional **E**ncoder **R**epresentations from **T**ransformers* [38]. It is capable of pre-training deep *bidirectional representations* from unlabeled text by jointly incorporating both left and right context of a sentence in all layers [38]. In addition, BERT has a fixed vocabulary of 30,000 tokens and can be fine-tuned for various NLP tasks such as question answering, NER, classification, Masked Language Modeling, next sentence prediction, etc [38].

Figure 3.2 illustrates what is meant by bi-directionality. In the example given,

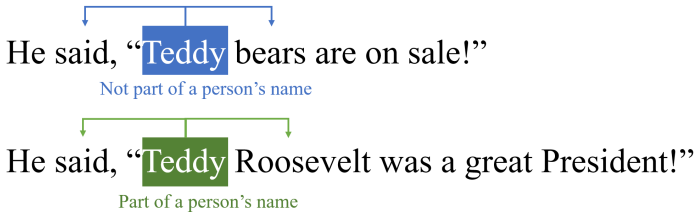


Figure 3.2: Example explaining the meaning of bidirectional [105]

LMs can predict the next word, and hence having information regarding what comes before and after the word *Teddy* will help with the prediction. If we know that the word following *Teddy* is *Roosevelt*, then the context is about the former president of the United States as compared to a teddy bear.

A simple example of transformers at work is shown in Figure 3.3, where it is being used for a language translation task [104]. It primarily consists of two components, namely an *encoder stack* and a *decoder stack*, where these stacks can consist of multiple encoders and decoders. The structures of the encoder layers are identical, however, they do not share the same weights [104].

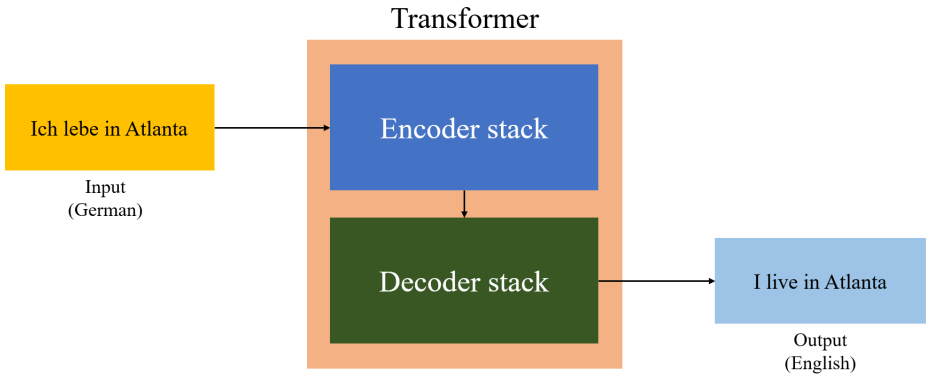


Figure 3.3: Transformer Architecture showcasing a language translation task[104]

Figure 3.4 takes us one step deeper into encoders and decoders showing how the layers come together to form an encoder or decoder block [104]. The encoder block can be broken down into *self-attention* and *feed forward neural network* layers. The *self-attention* layer encodes the different tokens in an input sequence while looking at other words in the sequence to make sure that the context is accounted for [104].

The output of the self-attention layer is then passed on to the feed forward NN. The decoder has the same layers as the encoder except for an extra *encoder-decoder attention* layer which helps with focusing on the appropriate part of the input sequence [104].

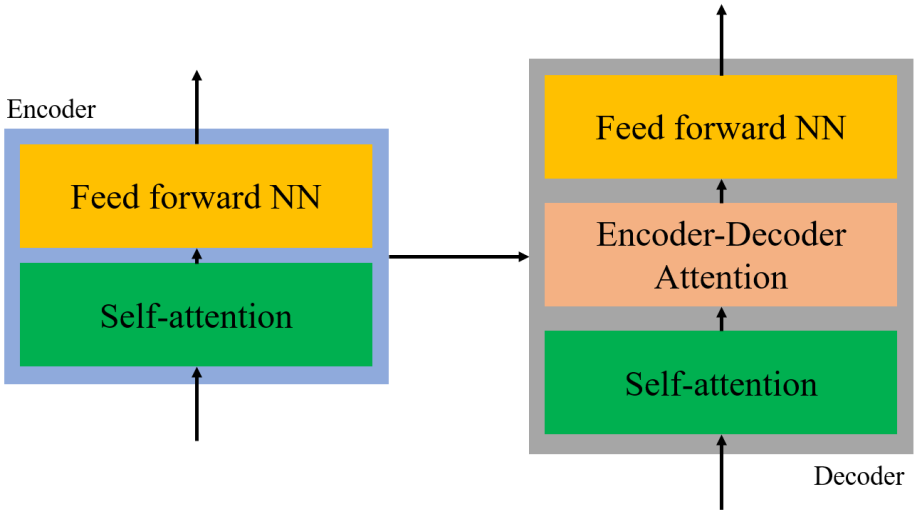


Figure 3.4: Transformer detailed architecture [104]

BERT uses only encoder layers and has two variations (Figure 3.5) [37], [38], [104]:

1. **BERT$_{\mathbf{BASE}}$**: contains 12 encoder blocks with a hidden size of 768, and 12 self-attention heads (total of 110 M parameters)

2. **BERT$_{\mathbf{LARGE}}$**: contains 24 encoder blocks with a hidden size of 1024, and 16 self-attention heads (total of 340M parameters)

Both BERT variants come in *cased* and *uncased* versions (Appendix C, Figure C.4). In the case of the *cased* model, the input text stays the same (containing both upper and lowercase letters). However, the text is lower-cased before being tokenized by a WordPiece tokenizer in the *uncased* model. Hence, there are four variants of BERT in total, as described below:

1. **BERT$_{\mathbf{LU}}$**: BERT large model trained on lower-cased English text

2. **BERT$_{\mathbf{LC}}$**: BERT large model trained on cased English text

3. **BERT$_{\text{BU}}$**: BERT base model trained on lower-cased English text

4. **BERT$_{\text{BC}}$**: BERT base model trained on cased English text

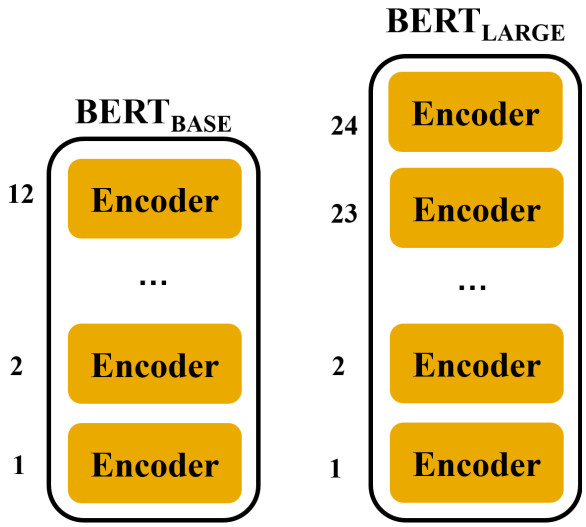


Figure 3.5: BERT$_{\text{BASE}}$ and BERT$_{\text{LARGE}}$ [104]

As mentioned previously, BERT is pre-trained on Book Corpus and English Wikipedia in a semi-supervised way (Figure 3.6) [38]. It is pre-trained to perform two tasks, namely, Masked Language Modeling (MLM) and Next Sentence Prediction (NSP) which are described below.

Masked Language Model (MLM): Due to BERT's bidirectional nature, each word would be capable of indirectly "seeing itself" and hence making the task of predicting the next word in a multi-layered setup trivial [38]. In order to counteract this pitfall, a certain percentage of the input tokens are *masked* which are then predicted using the LM. BERT masks 15% of the WordPiece tokens in an input sequence at random [38].

Next Sentence Prediction (NSP): BERT is pre-trained to predict whether one sentence directly follows the sentence prior to it. The LM was trained for this task by taking two sentences (A and B) into account where 50% of the time B was actually the next sentence and the rest of the times it was not. This task is crucial when it

comes to tasks such as Question Answering (QA) and Natural Language Inference (NLI) [38].

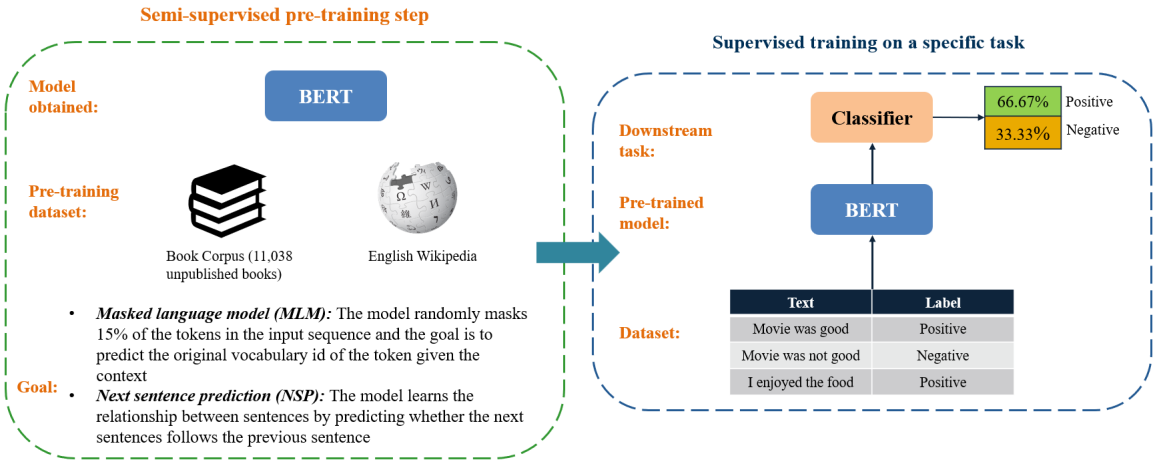


Figure 3.6: Pre-training and fine-tuning BERT language model [38], [104]

As a transformer-based LM, BERT uses a multi-headed attention mechanism (computing multiple attention vectors in parallel) to learn the complex structure and non-sequential context in language [37]. Each attention head learns to focus on particular syntactic relations [106]. To help readers develop intuition, the following three matrices are used to calculate attention using Equation 3.1 [37].

1. **Query (Q)**: Current position in the sequence
2. **Key (K)**: Relevance of each token in the sequence to the query
3. **Value (V)**: "Context-less" meaning of input tokens

$$Attention(Q, K, V) = softmax(\frac{Q * K^T}{\sqrt{(d_k)}}) * V \tag{3.1}$$

Where d_k = dimensionality of the key vectors; its square root is used for normalization of the attention scores

Appendix C (Figure C.14, Figure C.15, Figure C.16) provides a comprehensive illustration of the matrix multiplication process involved in computing self-attention.

Multi-headed attention scores can be calculated by computing multiple attention vectors in parallel using Equation 3.2 [37].

$$MultiHead(Q, K, V) = Concat(head_1, \ldots, head_h) * W^O \tag{3.2}$$

Where $head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$; W^O = learnable weight matrix

Figure 3.7: The 9th attention head of layer 3 in the encoder of BERT$_{BASE-UNCASED}$ illustrates how the query (Q) for the word "*record*" attends to the concept of "*what should be recorded*" in BERT's internal representation of the requirement,"*Each cockpit voice recorder shall record voice communications transmitted from or received in the airplane by radio.*"

Figure 3.7 depicts the Query (Q), Key (K), and Value (V) vectors as vertical bands, with the intensity of each band corresponding to its magnitude. The lines connecting these bands are weighted based on the attention between the tokens. Figure 3.7 shows the inner working of the 3rd encoder's 9th attention head which captures how the query vector "*record*" focuses on "*what should be recorded*" in BERT's internal representation of the requirement,*"Each cockpit voice recorder shall record voice com-*

munications transmitted from or received in the airplane by radio." This behavior exhibited by the attention heads occurs based solely on the self-supervised pre-training of BERT. Additional information regarding the matrix multiplications utilized in computing multi-headed attention is provided in Appendix C (Figure C.17).

BERT LM expects the input sequence to be in a certain format given the tasks it is pre-trained on (MLM and NSP). Details about the input format are discussed below:

- **Use of special tokens :**
  - **[CLS] :** This special token is added to the beginning of each sequence. The final hidden state of this token is a pooled output of all tokens and is used for sequence classification tasks [38], [66].
  - **[SEP] :** This special token is used to separate one sequence from the next [38].
  - **[PAD] :** Usually, the length of input sequences is set to a specified maximum number of words/tokens. This maximum length is decided upon after examining the distribution of lengths of all the sequences in the corpus. If any sequence is less than this maximum length, it is padded with [PAD] special tokens till the sequence length equals the set maximum length. Sequences that are longer than the specified maximum length are truncated [38].
- **BERT Embeddings :** Tokens in sequences need to be represented in a vectorized form (called embeddings) to serve as an input to BERT LM. Final BERT embeddings are a summation of position embeddings, segment embeddings, and token embeddings [38].

The use of special tokens and BERT embeddings is illustrated in Figure 3.8.

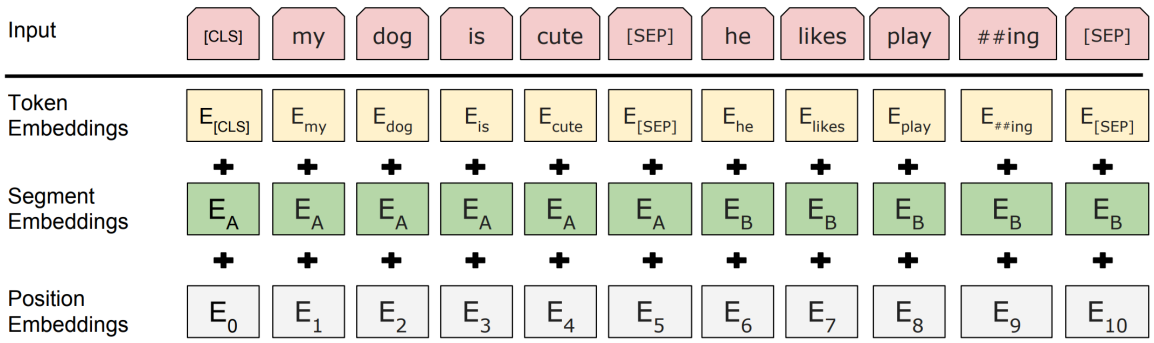


Figure 3.8: BERT embeddings and use of special tokens [38]

While BERT is pre-trained on two tasks, it can be fine-tuned to perform multiple downstream tasks such as NER, POS tagging, sentence classification and so on, hence paving the way for transfer learning. There are minimal differences between the pre-trained and fine-tuned BERT architectures, as shown in Figure 3.6 and Figure 3.9 [38]. Pre-trained parameters are used to initialize the BERT LM and these parameters are then fine-tuned with the help of labeled data for various downstream tasks [38]. Hence, the use of different types of labeled data will lead to different language models even though they were initialized with the same parameters [38].

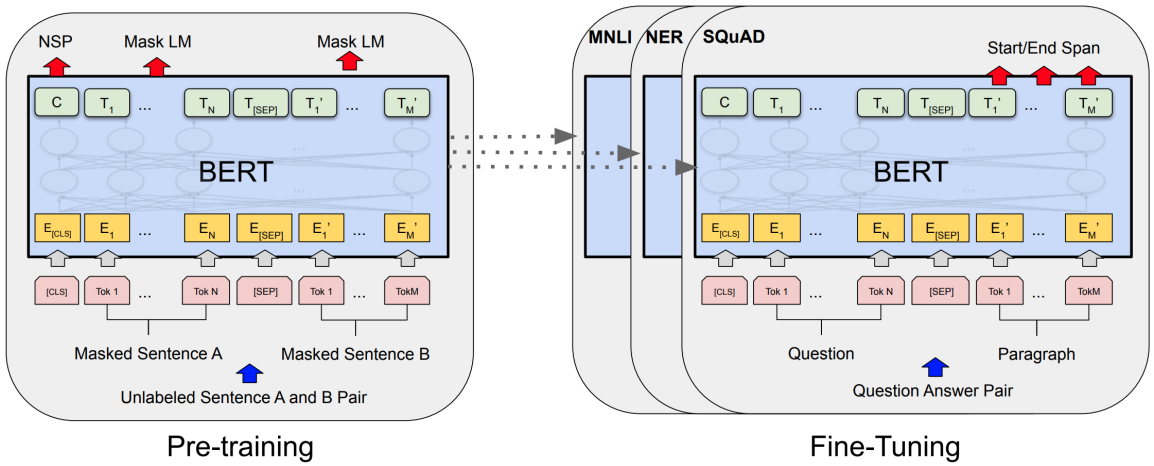


Figure 3.9: Pre-training and fine-tuning BERT LM [38]

To reiterate, BERT LM needs fine-tuning in order to be able to perform NER, such as organizations (ORG), locations (LOC), names of people (PER), and miscellaneous (MISC). There do exist fine-tuned models which are capable of performing NER tasks,

however, these models are trained on entity-annotated news articles from a specific time-frame [38], [107]. Hence they do not generalize when it comes to the aerospace domain. Table 3.2 shows the NE categories that a BERT NER model fine-tuned on the CoNLL-2003 Named Entity Recognition dataset (English) can identify [38], [107].

Table 3.2: Named Entity Recognition by $\text{BERT}_{\text{BASE}}$ [38], [107]

Category/Class	Symbol	Examples
Organizations	ORG	Apple, Google, USAF
Location	LOC	United States, Georgia, Atlanta
Person	PER	Timothy, William, Clara
Miscellaneous	MISC	Guidance and navigation system, etc.

The CoNLL-2003 Named Entity Recognition (NER) dataset is a widely used benchmark for evaluating NER models [107]. This English language dataset comprises news articles sourced from Reuters and was manually annotated by researchers at the University of Antwerp. For a comprehensive overview of the dataset, please refer to Table 3.3.

Table 3.3: CoNLL-2003 Language-Independent Named Entity Recognition dataset [107] showing the number of sentences and tokens included into the training, development, and test set.

English data	Sentences	Tokens	LOC	MISC	ORG	PER
Training set	14987	203621	7140	3438	6321	6600
Development set	3466	51362	1837	922	1341	1842
Test set	3684	46435	1668	702	1661	1617

Figure 3.10 shows the results obtained by applying fine-tuned BERT NER model on general-domain text (Example 1) and on aerospace-specific text (Examples 2, 3, and 4). The LM is able to identify the ORG (Delta Airlines) and LOC (Atlanta, GA) with very high confidence in Example 1. However, when used on aerospace-specific text, the same LM is unable to identify terms such as AoA (Example 2) and instead breaks it into two different sub-words 'A' and 'oA' which is not helpful. In example 3, the model is unable to identify terms like ETOPS and resources such as Part 121 and

Part 135 correctly. In addition, ISO 10303-233:2012 (from Example 4) was identified as MISC even though it is a resource.

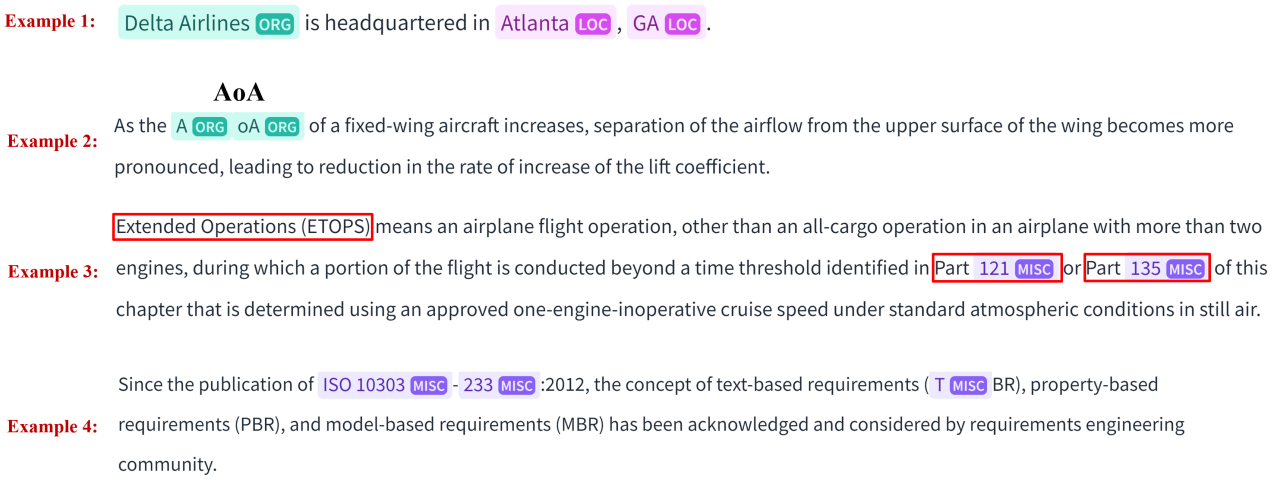


Figure 3.10: $\text{BERT}_{\text{BASE}}$ NER language model applied to general-domain and aerospace texts

These examples serve to emphasize the importance of employing transfer learning, specifically in fine-tuning BERT LM for NER in the aerospace domain (Figure 3.11). LLMs are typically composed of two parts - the Body and the task-specific head. The Body is capable of acquiring high-level features from the source domain, which can be fine-tuned to function in the target domain (aerospace in this context) with a small annotated dataset. Given that the LLMs have already been pre-trained on large English text corpora, it is reasonable to assume that fine-tuning can be accomplished with a small annotated NE corpus in aerospace, which is commonplace in the requirements engineering community due to the absence of significant annotated datasets. By doing so, the identification and extraction of terms for creating a glossary can be facilitated, ultimately enhancing the quality and comprehensibility of aerospace requirements [69].

This leads to the following research question:

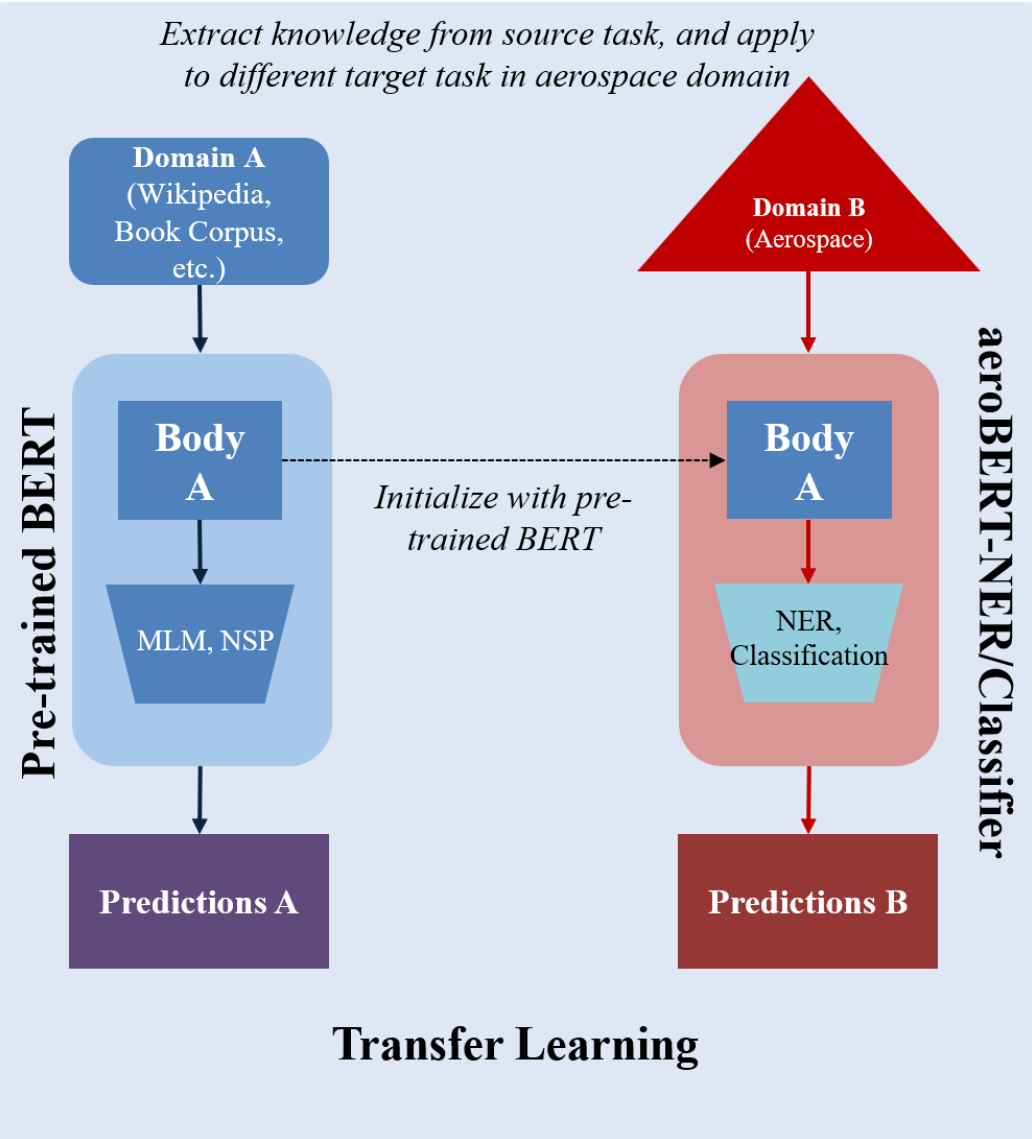


Figure 3.11: BERT LM can be fine-tuned for tasks such as NER, Classification, etc. in the aerospace domain despite having been pre-trained on general English corpora. Due to the lack of large annotated datasets in the aerospace requirements engineering domain, transfer learning was chosen as the path forward.

Research Question 1.1

How can we fine-tune the BERT language model to identify named-entities specific to the aerospace domain?

Being able to identify NEs will help with effective communication between different stakeholders due to the use of consistent language [69]. In addition, it will make the task of finding resources, system names, quantities, etc. being used in a system description much easier [69].

Traditional text extraction tools and the LMs trained on non-aerospace text lead to poor recall when applied to aerospace requirements [69] which translates into their inability to identify relevant terms. However, as a pre-requisite to being able to fine-tune BERT LM for NER identification for the aerospace domain, a corpus with annotated aerospace NEs is required. Such a corpus does not exist and will be created as a part of this dissertation. This will involve the collection of texts from the aerospace domain followed by an annotation task which will be discussed in Chapter 4.

The identification of NEs is difficult for two main reasons, namely *Segmentation* and *Type Ambiguity* [54] which are discussed below.

Segmentation: Every token in a sentence has one associated POS, however, this is not the case when it comes to NEs. In addition, multiple tokens might be one NE. Hence, it is helpful to address this issue during the text annotation phase by using BIO tagging scheme which is a standard for NER [38], [68] [Table 3.4]. For the sentence "I like Atlanta.", the POS and BIO tagging is shown in Table 3.5.

Table 3.4: BIO tagging notation [38], [68]

NE tagging	Meaning of tag
B	beginning of entity
I	inside an entity
O	outside an entity

Table 3.5: BIO tagging NE annotation task [38], [68]

Token	POS tag	NER tag
I	PRON	O
like	VERB	O
Atlanta	PROPN	B-LOC

Type Ambiguity: This type of ambiguity arises when there can be multiple meanings for a given token. Let's try to understand this by the help of two sentences:

Sentence A: *Washington* was the first president of the USA.

Sentence B: The airplane is headed to *Washington*.

The word *Washington* occurs in both Sentences A and B; however, they mean different things given the different contexts. *Washington* in Sentence A refers to a person (PER), whereas it is a location (LOC) in Sentence B. This ambiguity stresses the fact that context is crucial for understanding a sentence. BERT is capable of understanding this distinction because of its bidirectional property and the use of self-attention mechanism.

The above discussion leads to the first hypothesis:

Hypothesis 1.1: *If an annotated aerospace NE corpus is used to fine-tune BERT, then we will be able to identify NEs specific to the domain.*

To validate Hypothesis 1.1, a NER LM with aerospace domain knowledge needs to be developed. This new model will be named **aeroBERT-NER**. The performance of $\text{BERT}_{\text{BASE}}$ NER LM will be compared with aeroBERT-NER's performance on aerospace text for validation of the hypothesis. An overview of Experiment 1.1 which will facilitate the testing of Hypothesis 1.1 is provided below.

Experiment 1.1: *Establish the capability to identify aerospace-specific NEs from aerospace requirements and text. This capability or model can be validated by applying it to test aerospace requirements and measuring the performance by examining the following:*

1. Ability to extract NEs from aerospace requirements in a reliable and repeatable manner.

2. Comparing the performance of $\text{BERT}_{\text{BASE}}$ NER and aeroBERT-NER on previously unseen text to check if the incorporation of aerospace domain knowledge during the fine-tuning step led to improvements.

If the above capabilities are realized then it can be concluded that the fine-tuning of $\text{BERT}_{\text{BASE}}$ LM for NER with annotated aerospace NE corpus improved the performance of the model on aerospace text, hence, substantiating Hypothesis 1.1. The hypothesis will be rejected if the above capabilities can not be achieved.

The experimental steps for Experiment 1.1 are stated below:

- **Step 1:** An annotated aerospace NE corpus is obtained, which will serve as input for the fine-tuning process.

- **Step 2:** The following pre-processing tasks were performed:

  - The dataset was split into training and test sets to avoid data leakage
  - The maximum length of the input sequence was selected based on the distribution of lengths of the sequences in the training set
  - The input sequence was be tokenized and special tokens such as [CLS], [SEP], and [PAD] were added, as required
  - BERT embeddings were obtained

- **Step 3:** Full fine-tuning of BERT for NER was performed.

- **Step 4:** Metrics such as precision, recall, and F1 score were used to measure the performance of aeroBERT-NER against $\text{BERT}_{\text{BASE}}$ NER.

The confusion matrix is shown in Figure 3.12 and the equations for evaluation metrics (accuracy, precision, recall, and F1 score) are provided below:

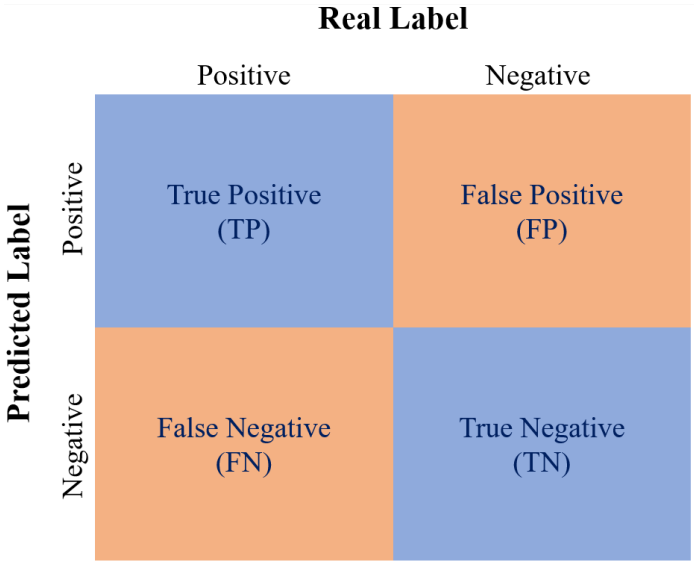


Figure 3.12: Confusion Matrix

- **Accuracy**: Defined by the summation of true positives (TP) and true negatives (TN) divided by the total number of items as shown in Equation 3.3.

  Accuracy is not a good metric when dealing with an imbalanced class problem.

$$Accuracy = \frac{TP + TN}{Total\ number\ of\ items} \tag{3.3}$$

- **Precision**: Defined by the number of true positives (TP) divided by the summation of true positives and false positives (FP) (Equation 3.4). The higher the number of TPs, the higher the precision. The higher the number of FPs, the lower the precision.

$$Precision = \frac{TP}{TP + FP} \tag{3.4}$$

- **Recall**: Defined by the number of true positives (TP) divided by the summation of true positives and false negatives (FN) (Equation 3.5). The higher the number

of TPs, the higher the recall. The higher the number of FNs, the lower the recall.

$$Recall = \frac{TP}{TP + FN} \tag{3.5}$$

- **F1 Score:** This is the harmonic mean of precision of recall (Equation 3.6). A higher F1 score is considered good.

$$F1\ Score = \frac{2TP}{2TP + FP + FN} \tag{3.6}$$

3.3 Research Question 2: Identification of boilerplates

Gap #2 - lack of requirement analysis methods for the classification of aerospace requirements and the identification of appropriate boilerplate structures given a requirement type will be addressed by RQ 2. The end goal is to be able to convert NL requirements into standardized requirements paving the way for easier analysis and transition towards machine-readable requirements. The problem stated above warrants the following research question.

Research Question 2

How can NL requirements be converted into appropriate boilerplates of interest?

To answer the above question, we need to understand the NLP sub-tasks that can facilitate the conversion of NL requirements into standardized form. The sub-tasks are stated below:

1. **Classification of aerospace requirements**: Being able to classify requirements will help with finding textual patterns in a certain type of requirement, which can then be used for the construction of boilerplates [88]. For example, the textual patterns observed in design requirements might be different than those for data requirements, and so on.

2. **Identifying named entities (NEs) in aerospace requirements**: Being able to identify and extract NEs from aerospace requirements will help with accessing data within requirements rather than treating the statement as a standalone object [30]. The identification of NEs was addressed in RQ 1.1.

3. **POS tagging/Text chunking**: Information regarding the syntactic patterns present in aerospace requirements will be helpful for requirement standardization.

4. **Identification of textual patterns in requirements**: Checking for patterns (linguistic constructs) that occur in a substantial number of requirements will be crucial for identifying boilerplates [108], [109].

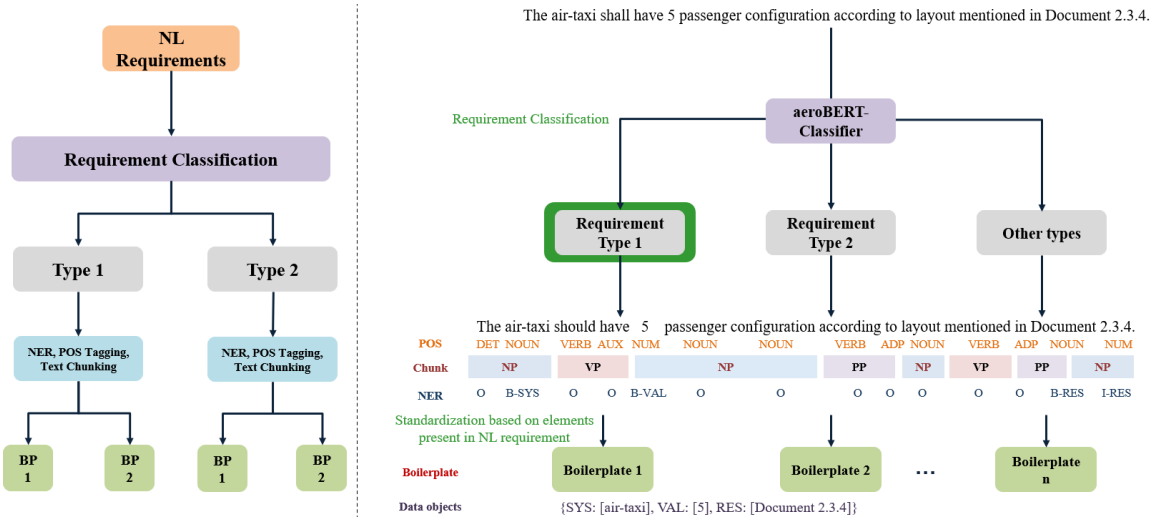


Figure 3.13: Research Question 2 flowchart

Figure 3.13 shows the steps that need to be followed for answering RQ 2. The process for the identification of boilerplates and requirement standardization begins with the classification of requirements into various types. After the classification, the POS tags, text chunks, and NEs associated with each token in the requirement sequences for each type of requirement are obtained. Based on the observed textual patterns, the conversion of requirement elements into data objects and boilerplates

can be developed for each type of requirement in a semi-automated way. It might be the case that more than one boilerplate patterns exist for each requirement type. In Figure 3.13, the raw NL requirement first passes through the classification algorithm and is classified as a Type 1 requirement. This requirement is then passed through a POS tagger, text chunker, and aeroBERT-NER and tags are assigned to each token. Based on the frequency of different linguistic patterns (patterns in the tags and NEs), boilerplates can be constructed in a semi-automated manner for various types of requirements in an agile manner. This methodology makes it faster and easier to boilerplates that are more tailored for use at a particular industry/organization as compared to Rupp's or EARS boilerplate structures.

The following discusses in more detail the first step to address RQ 2: requirement classification.

Text classification is widely used in various fields such as newsgroup filtering, sentiment classification, marketing, medical diagnosis, etc. [110], [111]. It has also proven to be critical even in the oil industry for the classification of failed occupational health control and for resolving accidents [112]. While using text classification for various tasks is not a new thing, there has been no study when it comes to the classification of aerospace requirements.

As discussed previously, BERT can be fine-tuned for text classification in a supervised way [38]. Text classification can be of two types, namely, *binary* and *multi-class* classification. Binary classification has two classes, whereas multi-class classification deals with multiple classes. For the problem at hand, there is an inclination towards multi-class classification given there are more than two types of aerospace requirements. Another way to classify classification algorithms is *hard* or *soft* classification. A hard classification algorithm assigns one class to each instance, whereas, a soft algorithm assigns probabilities of a requirement belonging to a certain class [110].

This leads to the first sub-question under RQ 2:

Research Question 2.1

How can BERT language model be fine-tuned for classifying aerospace requirements?

Fine-tuning BERT to classify aerospace requirements requires labeled requirements for the model to be trained on. The format of the training dataset is shown in Table 3.6. Once trained, this model can be used for predicting the class a certain requirement belongs to, based on the classes that the model is trained on.

Table 3.6: Dataset for requirements classification

Requirement	Label
The air-taxi should have 5-passenger configuration according to layout mentioned in Document 2.3.4.	Type 1
The measurement system shall include a FAA approved one-third octave band analysis system.	Type 2

This leads to Hypothesis 2.1:

Hypothesis 2.1: *If BERT is fine-tuned with pre-labeled aerospace requirements, then the classification of new requirements can be achieved by using the fine-tuned model.*

To validate Hypothesis 2.1, a classifier LM with aerospace domain knowledge needs to be developed. This new model will be called **aeroBERT-Classifier**. Accuracy, precision, recall, and F1 score metrics will be used for measuring the performance of the model to account for any imbalanced class issues. An overview of Experiment 2.1, which will facilitate the testing of Hypothesis 2.1, is provided below.

Experiment 2.1: *Establish the capability to classify aerospace requirements into various pre-defined classes in an automated manner.*

Hypothesis 2.1 will be substantiated upon the development of a version of BERT LM that is capable of classifying aerospace requirements into various pre-defined classes. The hypothesis will be rejected if the above capabilities can not be achieved.

The experimental steps for Experiment 2.1 are stated below:

- **Step 1:** Aerospace requirements are labeled based on the type of requirement, this serves as an input for the fine-tuning process.
- **Step 2:** WordPiece Tokenizer [38] will be used to tokenize the input sequence and special tokens such as [CLS], [SEP], and [PAD] will be added.
- **Step 3:** The dataset will be split into a training and test set. Stratified sampling might be used in case of an unbalanced class problem.
- **Step 4:** Full fine-tuning of BERT for requirement classification will be performed.
- **Step 5:** Metrics such as precision, recall, and F1 score will be used to measure the performance of aeroBERT-Classifier on the test set.

The next step is to develop the capability to tag every token in a sequence with its corresponding POS tag.

There are eight main parts of speech for English, namely, nouns, pronouns, adjectives, verbs, adverbs, prepositions, conjunctions, and interjection [113]. Flair [114], developed by researchers at Humboldt University in Berlin is one of the state-of-the-art models when it comes to sequence tagging. A subset of the different POS tags used by Flair model are listed in Table 3.7.

Figure 3.14 shows some sentences from the aerospace domain along with their associated POS tags. The Flair [73] model was used for the POS tagging task. This sequence tagging model will be used for POS tagging aerospace text, which will then be used to fine-tune BERT LM.

This leads to the second sub-question under RQ 2:

Research Question 2.2

How can the BERT language model be fine-tuned for POS tagging?

Table 3.7: POS tagging notation used by Flair [114]

POS Tag	Meaning of tag
NNP	Proper noun, singular
VBZ	Verb, 3rd person singular present
VBN	Verb, past participle
PRP$	Possessive pronoun
JJ	Adjective
NN	Noun, singular or mass
IN	Preposition or subordinating conjunction
DT	Determiner
CD	Cardinal number
MD	Modal
RBR	Adverb, comparative
TO	to
CC	Coordinating Conjunction

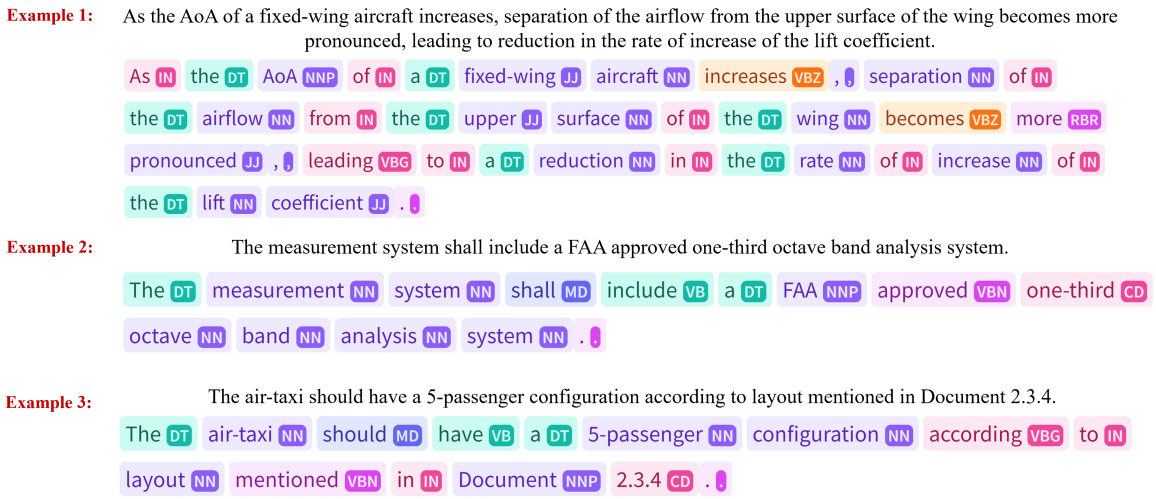


Figure 3.14: Example showing POS tagging by Flair [73]

The answer to RQ 2.2 will help map the sequence to their respective POS tags. The information provided by the POS tags can be helpful for improving syntactic parsing [54] that is helpful for the ordering of words in a sentence. In addition, POS tags can be helpful when it comes to measuring similarity/differences between sentences [54].

POS tagging has been done reliably (with high accuracy) by various supervised machine learning models such as Hidden Markov Models (HMM), RNNs, etc. [115]. However, certain pain points persist. Hence, while 85% of word types are unambigu-

ous, the 15% of the ambiguous word types are very commonly used and can have different meanings depending upon the context. Figure 3.15 shows the POS tags for a simple sentence "Alex will drive the car". The word "will" will always be an *auxiliary*. On the other hand, a *verb* would not follow the word "the". These are some of the ways in which POS tags are helpful in ordering words in a sentence.

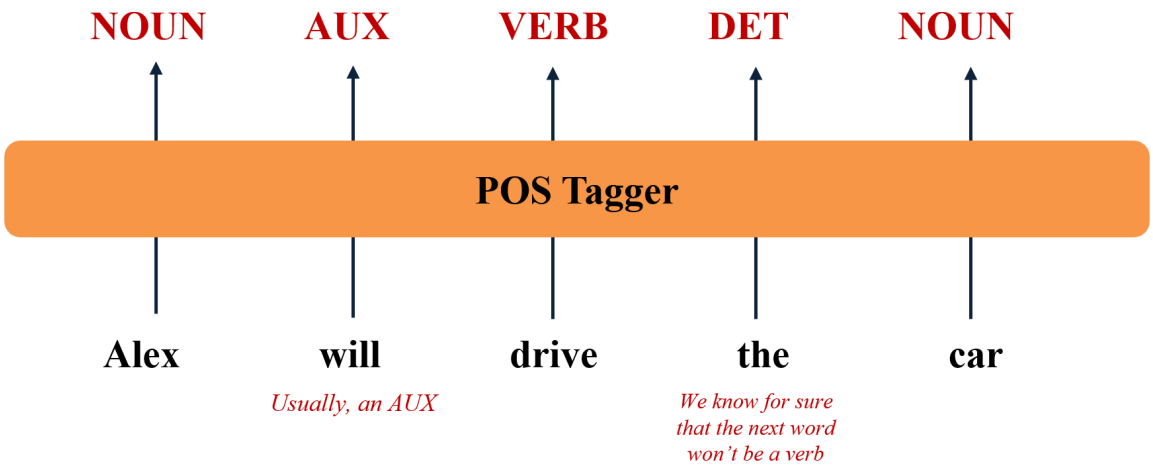


Figure 3.15: Example showing POS tags

- **Extra-curriculars took a back (ADJ) seat.**
- **Everyone will back (VERB) the bill.**
- **I was a teenager back (ADV) then.**

Figure 3.16: Example showing words with POS tags based on context [115]

Figure 3.16 shows that the word "back" can take up different POS tags based on the context [115], making POS tagging not very straightforward. Hence, the use of BERT LM for POS tagging can help with the problem at hand.

This leads to Hypothesis 2.2:

Hypothesis 2.2: *If an annotated POS tagged aerospace corpus is used to fine-tune BERT, then the identification of POS tags for tokens in NL aerospace requirements will be possible.*

To validate Hypothesis 2.2, a fine-tuned BERT POS tagger needs to be developed. This model will be called **aeroBERT-POStagger**. Accuracy, precision, recall, and F1 score metrics will be used for measuring the performance of the model.

An overview of Experiment 2.2, which will facilitate the testing of Hypothesis 2.2, is provided below.

Experiment 2.2: *Establish the capability to tag POS for aerospace requirements in an automated manner.*

Hypothesis 2.2 will be substantiated upon the development of a version of BERT LM which is capable of POS tagging tokens in aerospace requirement sequences. The hypothesis will be rejected if the above capabilities can not be achieved.

The detailed experimental steps for Experiment 2.2 are stated below:

- **Step 1:** The Flair sequence labeling model will be used for POS tagging of the aerospace corpus.
- **Step 2:** Special tokens such as [CLS], [SEP], and [PAD] will be added.
- **Step 3:** The dataset will be split into a training and testing set.
- **Step 4:** Full fine-tuning of BERT for POS tagging will be performed.
- **Step 5:** Metrics such as precision, recall, and F1 score will be used to measure the performance of aeroBERT-POStagger on the test set.

Following the capability to classify aerospace requirements and tag NEs and POS for tokens, requirements can be converted into a standardized form.

To reiterate the discussion from Section 2.3, there exists two well-known boilerplates such as Rupp's and EARS. However, these boilerplates can be restrictive and as such can contribute to ambiguities. In addition, Rupp's does not have blocks to include quantities, ranges of values, and references to external systems/devices/resources [18]. There is a need for customized boilerplates based on the contents of a NL requirement since requirement structures can vary significantly from one organization to another and even within an organization.

This leads to the third sub-question under RQ 2:

Research Question 2.3

How can requirements classification, NER, POS tagging, and text chunking be used for constructing boilerplates?

The answer to RQ 2.3 will help identify different boilerplate structures for non-standardized aerospace requirements. To that end a corpus-driven approach will be employed [108] for analyzing requirements to identify linguistic constructs. Semi-automatic bottom-up approach for defining elements of boilerplates using sequential text mining techniques have been proven to be useful [109]. Warnier et al. [109] used sequential data mining tool SDMC (Sequential Data Mining under Constraints) to find patterns in requirements text (in French) for two satellite projects. They ended up with ∼160,000 textual patterns initially and narrowed it down to 3,854 patterns after keeping the common patterns observed in requirements belonging to both the projects [109]. Finally, the number of observed linguistic patterns was reduced to 2,441 after discarding the patterns that were occurring in general-domain text such as newspaper articles [109].

In another study focused on requirement boilerplate structure, Kravari et al. [116] proposed that a requirement can be divided into three parts, namely, a prefix, a main part, and a suffix. While it is mandatory for a requirement to have one main part, it can have multiple prefixes (preconditions) or suffixes (additional information about a system's action, etc.). An ontology, as well as user input, was used for identifying boilerplate structures. While ontologies can be helpful, they are time-consuming to create and do not translate from one system to another.

According to another study by Ibrahim et al. [93], predefined boilerplate structures were helpful for novice systems engineers in writing requirements in a consistent and repeatable manner, hence reducing ambiguities and inconsistencies in NL requirements. The authors proposed boilerplates for both functional and non-functional requirements (performance, specific quality, and constraint). NL requirements per-

taining to an industrial case study (healthcare system MediNET) was used to validate the proposed boilerplate structures [93]. The study did not provide insights about how these boilerplate structures were decided upon and if/whether they can be tailored to other systems of interest, hence, making the methodology opaque.

All the studies discussed above, focus on rule-based approaches to developing boilerplates or requirement templates which reduces their viability in real-world industrial use cases [41]–[43], [117]. This is because the success of requirement boilerplates is heavily dependent on the consistency of requirements with the defined boilerplate structures. Oftentimes, NL requirements can have variations and hence might not perfectly match with the pre-defined boilerplates which is usually the case in large development projects (with less control over requirement authoring environments) [32], [41]–[44], [95], [117], [118] leading to lower accuracy. Hence, there is a need for an agile methodology for the creation of boilerplates/templates that are based on dynamically identified syntactic patterns in requirements, which is a more adaptive approach when compared to their rule-based counterparts.

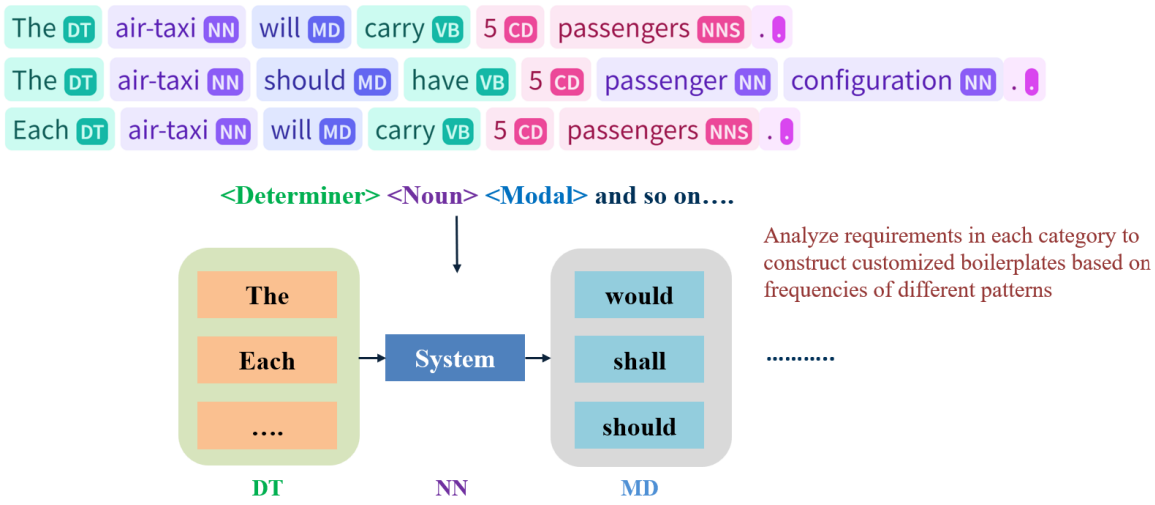


Figure 3.17: Boilerplate identification using linguistic constructs

Keeping in mind the need for an agile methodology for boilerplate creation, Figure 3.17 shows the process for the identification of linguistic constructs given different

variations of a requirement text. Despite the variations, the linguistic pattern ([Determiner][Noun][Modal][Verb][Cardinal], etc.) stays the same as shown in the example. Based on the frequency of different linguistic patterns along with information from the NER and text chunking models, boilerplates can be constructed for various types of requirements.

This leads to Hypothesis 2.3:

Hypothesis 2.3: *If sequential text mining techniques are employed to detect linguistic constructs, then boilerplates can be constructed based on the observed patterns in the requirements.*

To validate Hypothesis 2.3, sequential text mining techniques need to be developed to identify linguistic constructs which will aid in the boilerplate construction process.

Experiment 2.3: *Establish the capability to employ sequential text mining techniques for identification of linguistic patterns in a semi-automated manner.*

Hypothesis 2.3 will be substantiated upon the development of a methodology for the identification of linguistic patterns for construction of boilerplates for standardizing requirements. The hypothesis will be rejected if the above capabilities can not be achieved.

The detailed experimental steps for Experiment 2.3 are stated below:

- **Step 1:** aeroBERT-Classifier will be used for classifying aerospace requirements.
- **Step 2:** aeroBERT-NER will be used for identifying NEs in aerospace requirements.
- **Step 3:** aeroBERT-POStagger will be used for POS tagging the tokens in the requirements.
- **Step 4:** The frequencies of different linguistic patterns in the requirements text will be identified.

- **Step 5:** Based on the frequencies of different linguistic patterns, elements of the boilerplate and their order will be decided upon.
- **Step 6:** Steps 1-3 will be repeated for different types of requirements.
- **Step 7:** Iterations of the above steps will be carried out to make sure that the boilerplate construction system is robust.
- **Step 8:** The validity and accuracy of the boilerplates generated will be verified with the help of a Subject Matter Expert (SME).

3.4 Overarching Hypothesis

Aerospace requirements can be converted into semi-formal/semi-machine-readable formats if an NLP enabled methodology can be established to perform the following tasks collectively:

1. Automated extraction of named-entities (NEs) from aerospace requirements for creation of glossary leading to the standardization of terms being used by different stakeholders
2. Conversion of natural-language requirements into boilerplates in a semi-supervised manner which entails
   (a) Classifying of aerospace requirements into different types
   (b) Identifying NEs in aerospace requirements
   (c) Performing POS tagging for tokens present in a requirement
   (d) Creating of boilerplates from requirements by using sequential text mining for identifying linguistic constructs

3.5 Summary of Observations, gaps, and research questions

The summary of the observations, gaps, and research questions is provided in Figure 3.18.

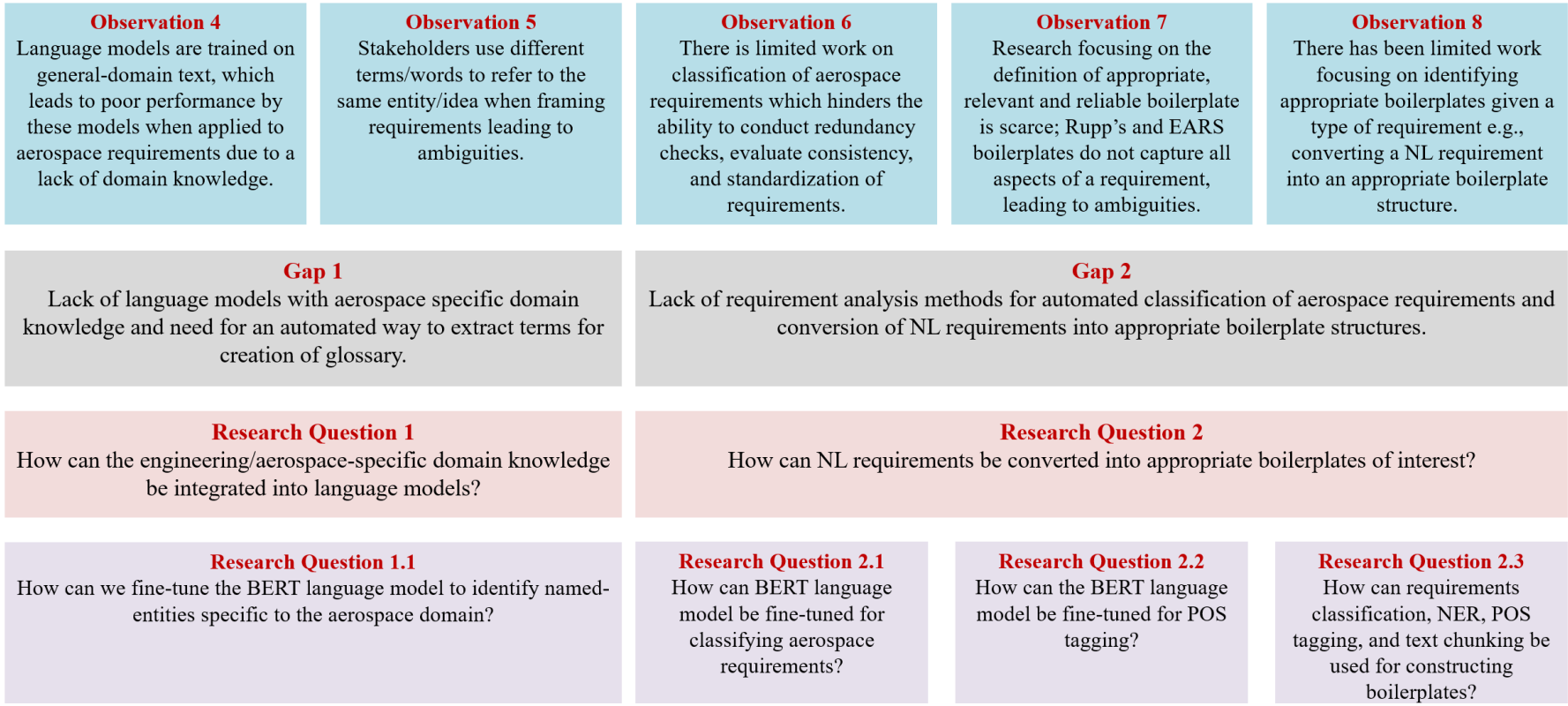


Figure 3.18: This figure presents a summary of the observations made, gaps identified, and research questions formulated. Specifically, Gap 1 was identified and Research Question 1 was formulated based on observations 4 and 5, while Gap 2 was identified, and Research Question 2 and its sub-research questions were formulated based on observations 6, 7, and 8.

CHAPTER 4

METHODOLOGY

The methodology to carry out the experiments prescribed in Chapter 3 is discussed in this chapter. It is divided into four steps, where each step details the process to carry out one experiment as shown in Table 4.1 and Figure 4.1.

Table 4.1: Overview of Proposed Methodology

Step	Associated RQ
0	Creation of an annotated aerospace corpus
1	RQ 1.1: Fine-tuning of BERT for aerospace NER
2	RQ 2.1: Fine-tuning of BERT for classification of aerospace requirements
3	RQ 2.2: Fine-tuning of BERT for POS tagging
4	RQ 2.3: Creation of aerospace requirement boilerplates

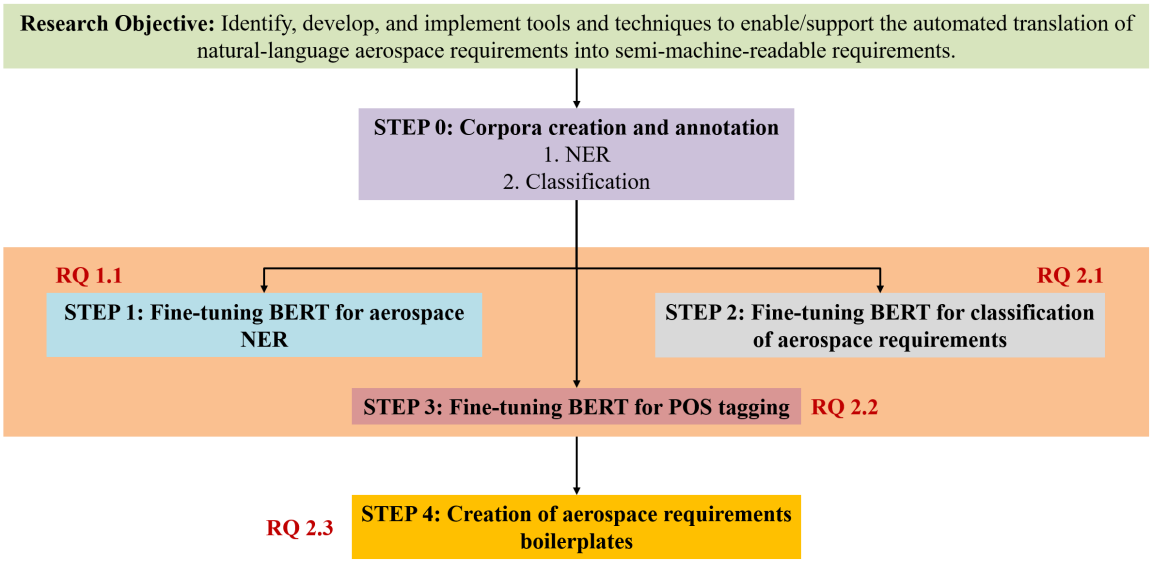


Figure 4.1: Overview of Proposed Methodology

4.1 Step 0: Development of an annotated aerospace corpus

The creation of an annotated aerospace corpus is of paramount importance because it is an input to the many steps to be carried out as part of this methodology. The first step towards creating a corpus is collecting aerospace-domain texts. Since BERT is pre-trained on English Wikipedia and BookCorpus, it should already have some information about the aerospace domain (articles about different aircraft, aviation agencies, etc.). Hence, the effort will be focused on collecting scientific aerospace texts to augment the context.

Machine Learning and Deep Learning models perform best when the training and test sets are of similar nature. As a result, the ideal dataset to train and test the LMs on would be a corpus that contains only aerospace requirements. However, this poses a problem since aerospace requirements are almost always proprietary and are not available in the open-source domain. An alternative approach is to use aerospace texts available in the open-source domain such as research papers (excluding figures, tables, etc.), aviation news articles, and parts of Title 14 of the Code of Federal Regulations (CFRs), which deal with certification requirements for new aircraft designs. Some of the 14 CFR parts are shown in Table 4.2 (this is not an exhaustive list).

Table 4.2: Title 14 CFR Parts [119]

Part	Subject
Part 1	Definitions and Abbreviations
Part 21	Certification Procedures for Products and Parts
Part 23	Airworthiness Standards: Normal, Utility, Acrobatic and Commuter Airplanes
Part 27	Airworthiness Standards: Normal Category Rotorcraft
Part 33	Airworthiness Standards: Aircraft Engines
Part 36	Noise Standards: Aircraft Type and Airworthiness Certification
Part 39	Airworthiness Directives

To reiterate, three types of aerospace text, namely, technical aerospace texts such as research papers, more general aerospace texts such as aviation news, and lastly certification requirements from Title 14 CFR are used to create the corpus. The

following steps are proposed to develop an annotated aerospace corpus:

- **Step 1:** Aerospace texts are collected for the creation of the corpora.
- **Step 2:** The collected texts are pre-processed to remove equations, etc., and to get the text into the correct format for the next step.
- **Step 3:** Annotation criteria for different downstream tasks are decided upon, for example, the types of NEs that are of interest, types of requirements, and type of POS tags to be considered. This is likely to be an iterative process.
- **Step 4:** Annotation task is carried out in a semi-automated way in this step using the Python language and its packages such as NLTK, spaCy, etc.:
  - BIO tagging notation is used for tagging NEs
  - Flair [73] is used for POS tagging
  - Different types of aerospace requirements are selected and labeled for the classification task

The Python programming language is used for all the coding tasks required since it is open-source and highly customizable for various tasks.

4.2 Step 1: Fine-tuning BERT for aerospace NER (aeroBERT-NER)

This section presents the methodology to carry out Experiment 1.1 and addresses RQ 1.1.

Figure 4.2 describes the experimental steps. The experiment begins with using the annotated NE text corpus from Step 0 as an input for the fine-tuning process. Various pre-processing steps are carried out, such as splitting the dataset into training and test sets, deciding on the maximum length of the input sequences, and obtaining the BERT embeddings. The pre-processed training set is used to fine-tune the BERT

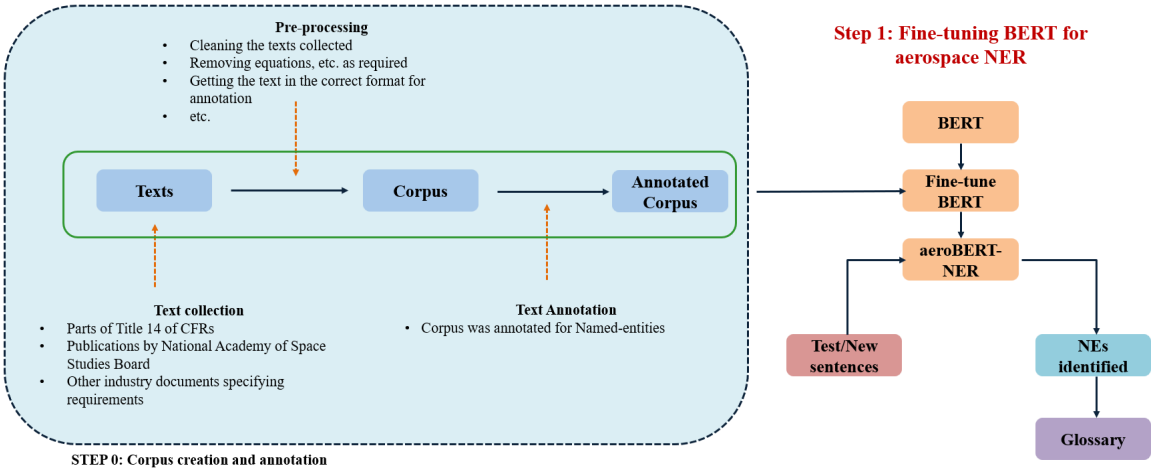


Figure 4.2: Methodology for obtaining aeroBERT-NER

parameters for NER. Evaluation metrics such as precision, recall, and F1 score are used to evaluate the model performance on a validation and test sets. In addition, the performance of aeroBERT-NER [120] is compared with that of $\text{BERT}_{\text{BASE}}$ NER on an aerospace requirements test set. Various iterations of the model training/testing are carried out to make sure that the results are robust and reliable.

4.3 Step 2: Fine-tuning BERT for aerospace requirement classification (aeroBERT-Classifier)

This section presents the methodology to carry out Experiment 2.1 and addresses RQ 2.1.

Figure 4.3 illustrates the experimental steps for Experiment 2.1, which aims at developing a classifier for the automated classification of aerospace requirements. The methodology starts with labeled aerospace requirements, which are divided into training and test sets. The labeled training set of aerospace requirements is used for fine-tuning the parameters of BERT LM for the classification task. Evaluation metrics such as accuracy, precision, recall, and F1 score are used to evaluate the model performance on the test set to account for the imbalanced dataset. Various iterations of the model training/testing are carried out to make sure that the results

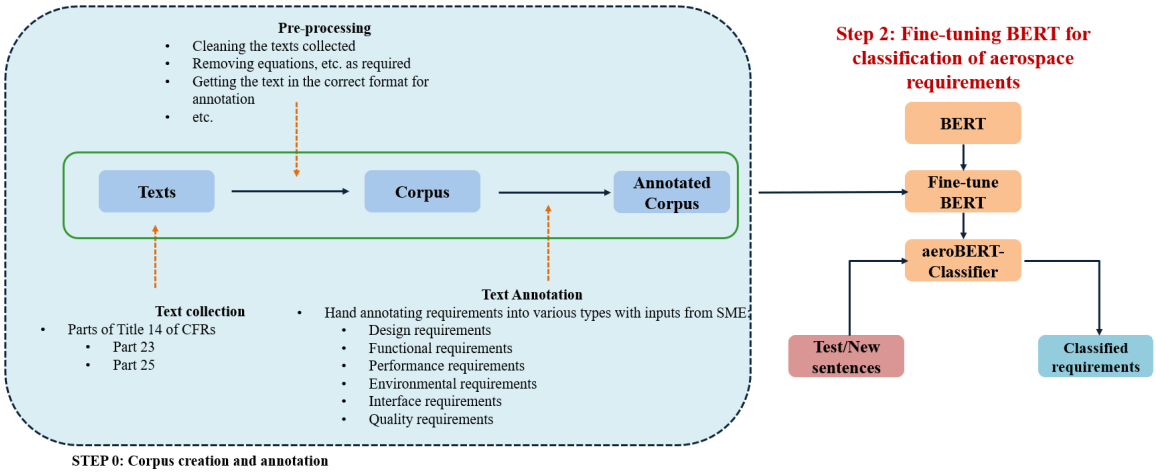


Figure 4.3: Methodology for obtaining aeroBERT-Classifier

are robust and reliable.

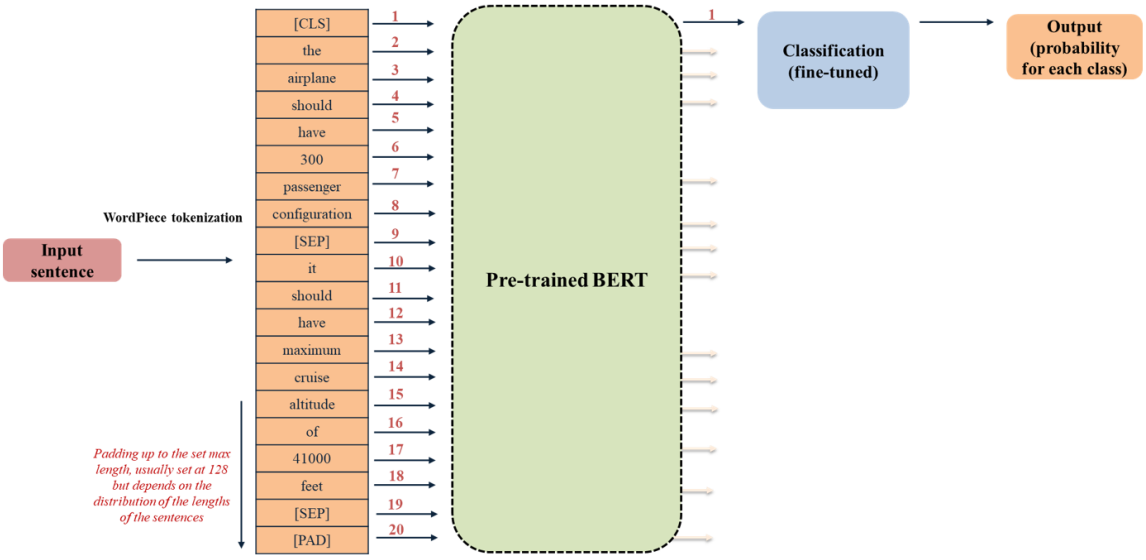


Figure 4.4: aeroBERT-classifier flowchart for requirement classification [38], [83]

Figure 4.4 provides a detailed view of the classification algorithm. The process starts with an input sequence/sentence, which is tokenized using a WordPiece tokenizer. Special tokens such as [CLS], [SEP], and [PAD] are added to the tokenized sequence to mark the beginning, end and padding token. The [PAD] token is used to pad the sequence up to the set maximum length, which is dependent upon the distribution of length of sequences. This pre-processed sequence serves as an input to

the fine-tuning process. Only the [CLS] token is used for the classification task since it is a pooled output of all tokens. An activation function is used for outputting the probabilities of a given requirement belonging to each of the pre-defined classes. The requirement is classified into the class which has the highest probability.

4.4 Step 3: Fine-tuning BERT for POS tagging of aerospace text (aeroBERT-POStagger)

This section presents the methodology to carry out Experiment 2.2 and addresses RQ 2.2.

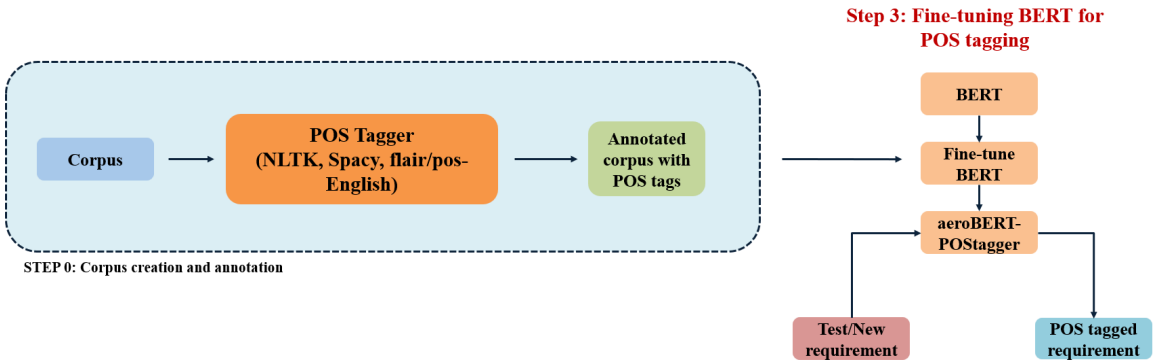


Figure 4.5: Methodology for obtaining aeroBERT-POStagger

Figure 4.5 illustrates the various steps to carry out Experiment 2.2. The annotated corpus with POS tags (from Step 0) serves as input to fine-tune BERT for POS tagging. Evaluation metrics such as precision, recall, and F1 score are used to evaluate the model performance on the test set. The fine-tuned model receives the test set (which it has not seen before) and performs the POS tag predictions on it. Various iterations of the model training/testing are carried out to make sure that the results are robust and reliable.

4.5 Step 4: Creation of aerospace requirements boilerplates

Figure 4.6 lays out the steps to carry out Experiment 2.3, which contributes to the goal of converting NL aerospace requirements into standardized/machine-readable requirements. This step needs inputs from Steps 1, 2, and 3 (discussed previously). Given multiple requirements belonging to a certain type, the frequency of different linguistic patterns is identified. Based on the observed frequencies of different linguistic patterns, the ordering of elements for boilerplates is identified. This process is repeated for different types of requirements. Lastly, the verification of the validity and accuracy of the proposed boilerplates is carried out by SMEs.

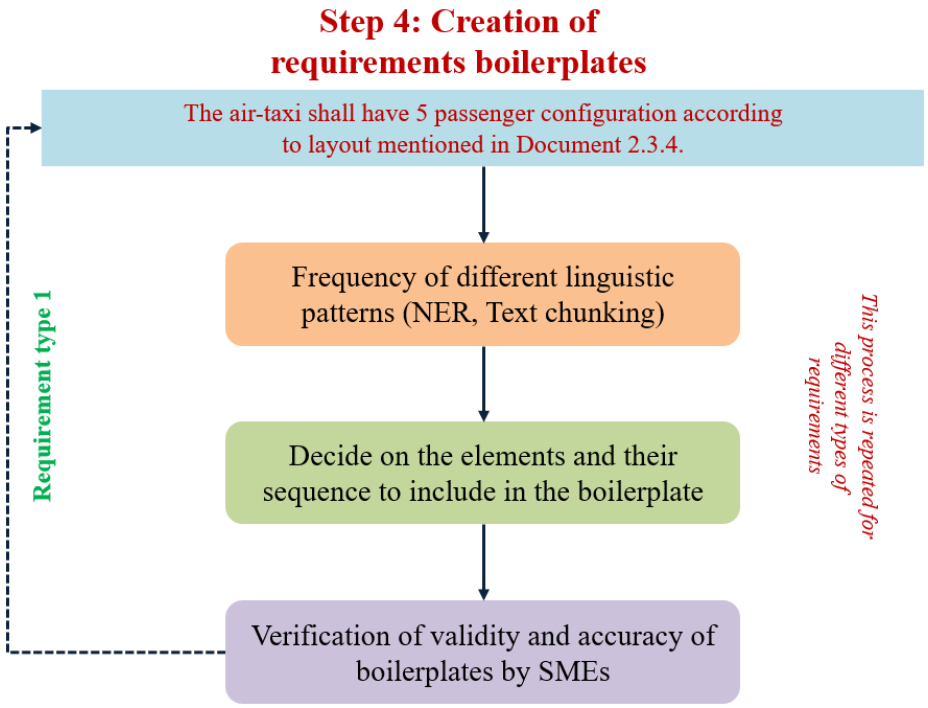


Figure 4.6: Methodology for aerospace requirement boilerplate construction

CHAPTER 5
IMPLEMENTATION

This chapter discusses the implementation of the methodology presented in the previous chapter.

5.1 Creation of annotated aerospace corpora

A LM such as BERT can be fine-tuned for various downstream tasks, such as NER, text classification, POS tagging, question answering, etc. Pre-trained parameters are used to initialize the BERT LM and these parameters are then fine-tuned with the help of labeled data for the downstream task. The use of different types of labeled data will lead to different models that perform different tasks even though they were initialized with the same parameters. For the purpose of this work, BERT LM is fine-tuned for three different downstream tasks, namely, NER, requirements classification, and POS tagging. As a result, three different annotated corpora need to be created, as discussed in the following sections.

5.1.1 Annotated corpus for aerospace Named-entity recognition (NER)

The creation of an annotated aerospace corpus is of paramount importance because it serves as a labeled dataset to fine-tune BERT for NER. The first step toward creating a corpus consists of collecting aerospace-domain texts. To that end, two types of aerospace texts are used to create the aerospace corpus for fine-tuning BERT: (1) general aerospace texts such as publications by the National Academy of Space Studies Board, and (2) certification requirements from Title 14 CFR. A list of all the documents used for creating the aerospace corpus for this research is provided in Table 5.1.

Table 5.1: Resources for creation of annotated aerospace NER corpus

Serial No.	Name of resource
1	Part 23: Airworthiness Standards: Normal, Utility, Acrobatic and Commuter Airplanes
2	Part 25: Airworthiness Standards: Transport Category Airplanes
3	Achieving Science with CubeSats (National Academy SSB Publication)
4	CubeSat Design Specification (CalPoly)

1432 sentences related to the aerospace domain were incorporated into the corpus. This was done as the annotation of the corpus is a time-consuming task, and hence a sufficient number of sentences were selected to demonstrate the methodology. For a detailed breakdown of the types of sentences included, please refer to Figure 5.1.

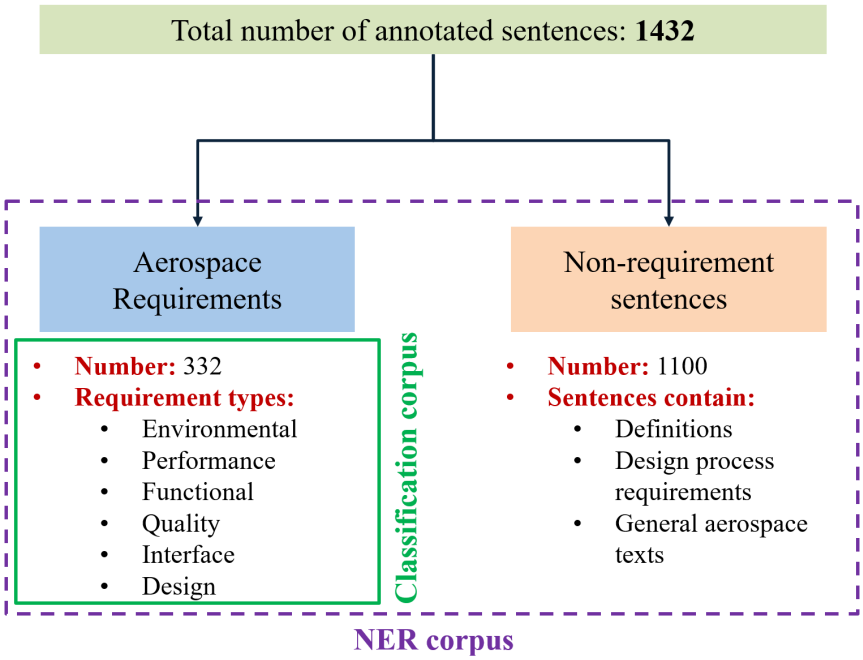


Figure 5.1: Types of sentences included in the aerospace corpus

These sentences were obtained by modifying the original text (when required) into a proper format for the purpose of corpus creation. For example, 14 CFR §23.1457(g) is shown in its original and modified form in Table 5.2. In this example, the original text was converted into three distinct requirements so that they can each be complete sentences.

In addition to modifying the original text (Table 5.2), figures, tables, and equations

Table 5.2: Text modification and requirement creation

14 CFR §23.1457(g)	Requirements created
Each recorder container must -	
(1) Be either bright orange or bright yellow;	**Requirement 1:** Each recorder container must be either bright orange or bright yellow.
(2) Have reflective tape affixed to its external surface to facilitate its location under water; and	**Requirement 2:** Each recorder container must have reflective tape affixed to its external surface to facilitate its location under water.
(3) Have an underwater locating device, when required by the operating rules of this chapter, on or adjacent to the container, which is secured in such manner that they are not likely to be separated during crash impact.	**Requirement 3:** Each recorder container must have an underwater locating device, when required by the operating rules of this chapter, on or adjacent to the container, which is secured in such manner that they are not likely to be separated during crash impact.

were discarded from the original text. Some other changes were made to certain entities, as shown in Table 5.3. The dots '.' which were not sentence endings were replaced with a '-' so as not to cause any confusion for the model which is pre-trained on general domain corpora. In addition, the symbol for section ('§'), was replaced with the word 'Section' to make it more intuitive for the model to learn patterns.

Table 5.3: Symbols that were modified for corpus creation

Original Symbol	Modified text/symbol	Example
§	Section	§25.531 → Section 25.531
§§	Sections	§§25.619 and 25.625 → Sections 25.619 and 25.625
Dot ('.') used in section numbers	Dash ('-')	Section 25.531 → Section 25-531

The aerospace corpus obtained was then annotated using the BIO tagging scheme (Table 3.5). For the purpose of aeroBERT-NER, five classes of NEs were identified based on their frequency of occurrence in aerospace texts (Table 5.4): System (SYS), Value (VAL), Date time (DATETIME), Organization (ORG), and Resource (RES).

After identifying the NE classes of interest, the aerospace corpus was annotated. NEs were identified in the corpus and added to distinct lookup *.txt* files for each class. For example, *auxiliary power unit* was added to the *systems.txt* file since it was identified as a system. A detailed chronology for the creation of lookup files is shown

Table 5.4: Types of named-entities identified

Category	NER Tags	Example
System	B-SYS, I-SYS	exhaust heat exchangers, powerplant, auxiliary power unit
Value	B-VAL, I-VAL	1.2 percent, 400 feet, 10 to 19 passengers
Date time	B-DATETIME, I-DATETIME	2013, 2019, May 11,1991
Organization	B-ORG, I-ORG	DOD, Ames Research Center, NOAA
Resource	B-RES, I-RES	Section 25-341, Sections 25-173 through 25-177, Part 23 subpart B

in Figure 5.2. These lookup files were then used for semi-automated NE annotation of the aerospace corpus.

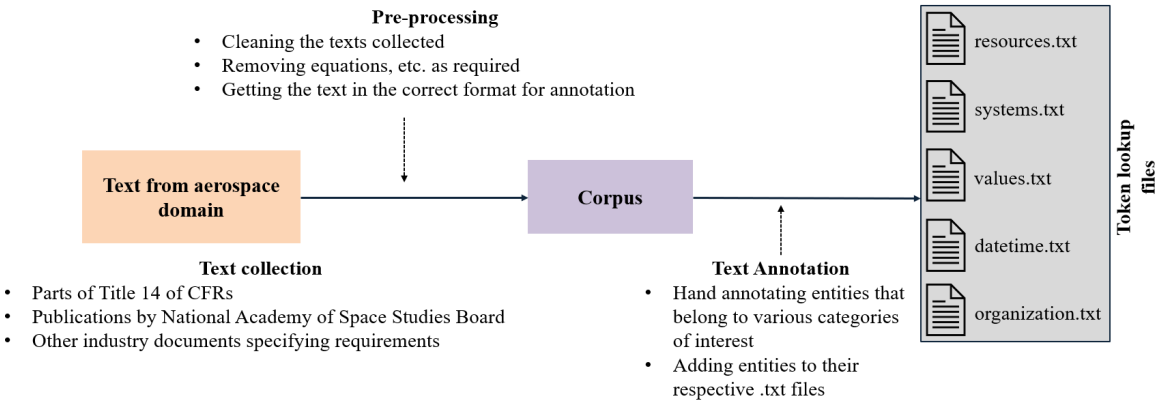
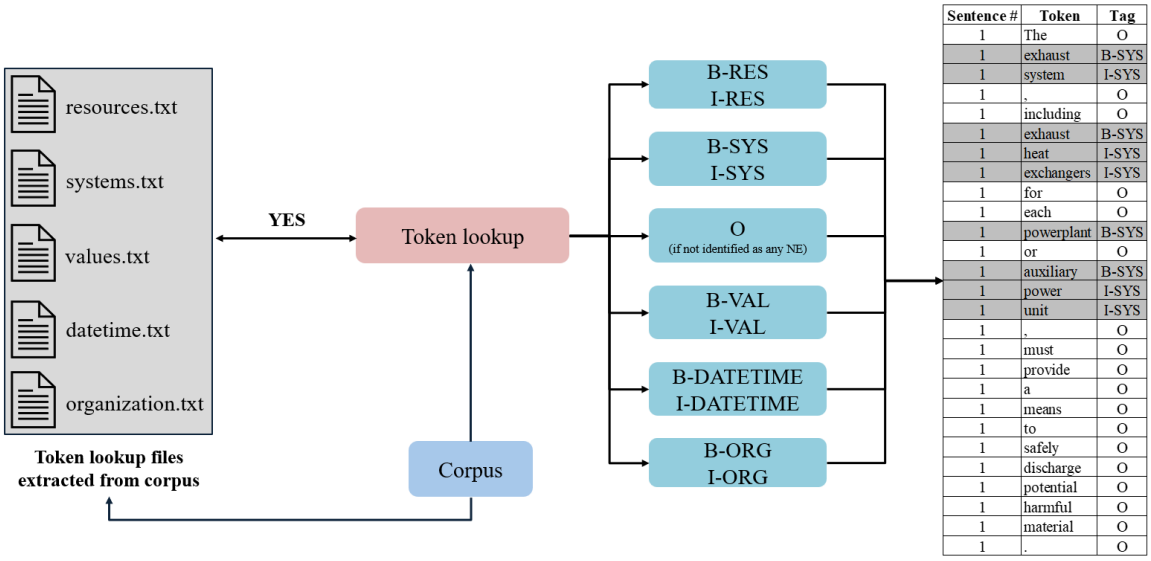


Figure 5.2: Flowchart showing creation of lookup files for NER annotation

For the NER annotation, if a token or sequence of tokens is found in the lookup *.txt* files, then it is tagged according to the name of the *.txt* file in which the token/tokens were found. For example, if *exhaust system* is found in the *systems.txt* file, then it will be tagged as a *system*. The flowchart detailing the NER annotations for the requirement "*The exhaust system, including exhaust heat exchangers for each powerplant or auxiliary power unit, must provide a means to safely discharge potential harmful material*", is shown in Figure 5.3. All the annotations were done according to the BIO tagging scheme provided in Table 3.5.

Lastly, Table 5.5 shows the number of times each NER tag occurs in the aerospace corpus created for aeroBERT-NER. System names (B-SYS, I-SYS) occur the most often, whereas DATETIME entities occur the least often in the corpus. The dataset for NER contains a total of 44,033 tokens.



Sentence #	Token	Tag
1	The	O
1	exhaust	B-SYS
1	system	I-SYS
1	,	O
1	including	O
1	exhaust	B-SYS
1	heat	I-SYS
1	exchangers	I-SYS
1	for	O
1	each	O
1	powerplant	B-SYS
1	or	O
1	auxiliary	B-SYS
1	power	I-SYS
1	unit	I-SYS
1	,	O
1	must	O
1	provide	O
1	a	O
1	means	O
1	to	O
1	safely	O
1	discharge	O
1	potential	O
1	harmful	O
1	material	O
1	.	O

Figure 5.3: Flowchart showing NER annotation methodology for the requirement "*The exhaust system, including exhaust heat exchangers for each powerplant or auxiliary power unit, must provide a means to safely discharge potential harmful material.*"

It took a total of four months to collect data and perform the initial annotation for Named Entity Recognition (NER). The annotation task was carried out by one human annotator who possessed knowledge of the aerospace domain. To ensure consistency, a second and third review of the annotation was conducted.

Table 5.5: NER tags and their counts in aerospace corpus for aeroBERT-NER

NER Tag	Description	Count
O	Tokens that are not identified as any NE	37686
B-SYS	Beginning of a system NE	1915
I-SYS	Inside a system NE	1104
B-VAL	Beginning of a value NE	659
I-VAL	Inside a value NE	507
B-DATETIME	Beginning of a date time NE	147
I-DATETIME	Inside a date time NE	63
B-ORG	Beginning of a organization NE	302
I-ORG	Inside a organization NE	227
B-RES	Beginning of a resource NE	390
I-RES	Inside a resource NE	1033
Total	**-**	**44033**

5.1.2 Annotated corpus for aerospace requirements classification

Similarly, an annotated aerospace requirements corpus is a critical step since such a corpus is not readily available in the open-source domain and is required for fine-tuning BERT for requirements classification. Table 5.6 provides a list of the resources leveraged to obtain requirements for the purpose of creating a classification corpus, which is a subset of the corpus that was annotated for NER.

Table 5.6: Resources used for the creation of aerospace requirements classification corpus

Serial No.	Name of resource
1	Part 23: Airworthiness Standards: Normal, Utility, Acrobatic and Commuter Airplanes
2	Part 25: Airworthiness Standards: Transport Category Airplanes

The collected requirements corpus underwent the same pre-processing steps as shown in Table 5.2. This involved converting the original text (which often occurs in paragraph form) into distinct single-sentence requirements. In addition, certain symbols which occurred in the text were modified according to the scheme shown in Table 5.4. A total of 325 requirements were collected and added to the classification corpus.

After pre-processing the requirements text, the next step consisted of annotating the requirements based on their type/category of requirement. Requirements were classified into six categories, namely, Design, Functional, Performance, Interface, Environmental, and Quality. The definitions used for the requirement types/categories along with the examples are provided in Table 5.7. An SME was consulted to make sure that the requirements were annotated correctly into various categories. It is important to keep in mind that different companies/industries might have their own definitions for requirements specific to their domain.

As mentioned previously, the corpus includes a total of 325 aerospace requirements. 134 of these 325 requirements (41.2%) were annotated as Design requirements,

Table 5.7: Definitions used for labeling/annotating requirements [5], [7], [121]

Requirement Type	Definition
Design	Dictates "how" a system should be designed given certain technical standards and specifications; **Example:** Trim control systems must be designed to prevent creeping in flight.
Functional	Defines the functions that need to be performed by a system in order to accomplish the desired system functionality; **Example:** Each cockpit voice recorder shall record voice communications of flightcrew members on the flight deck.
Performance	Defines "how well" a system needs to perform a certain function; **Example:** The airplane must be free from flutter, control reversal, and divergence for any configuration and condition of operation.
Interface	Defines the interaction between systems [122]; **Example:** Each flight recorder shall be supplied with airspeed data.
Environmental	Defines the environment in which the system must function; **Example:** Each electrical and electronic system that performs a function, the failure of which would prevent the continued safe flight and landing of the airplane, must be designed and installed such that the function at the airplane level is not adversely affected during and after the time the airplane is exposed to the HIRF environment.
Quality	Describes the quality, reliability, consistency, availability, usability, maintainability, and materials and ingredients of a system [123]; **Example:** Internal panes must be made of nonsplintering material.

91 (28%) as Functional requirements, and 62 (19.1%) as Performance requirements. Figure 5.4 shows the counts for all the requirement types.

As seen in Figure 5.4, the dataset is skewed toward Design, Functional, and Performance requirements (in that order). Since the goal is to develop a LM that is capable of classifying requirements, a balanced dataset is desired, which is not the case here. As seen, there are not enough examples of Interface, Environment, and Quality requirements in the primary data source (Parts 23 and 25 of Title 14 of the Code of Federal Regulations (CFRs)). This can be due to the fact that Interface, Environment, and Quality requirements do not occur alongside the other types of

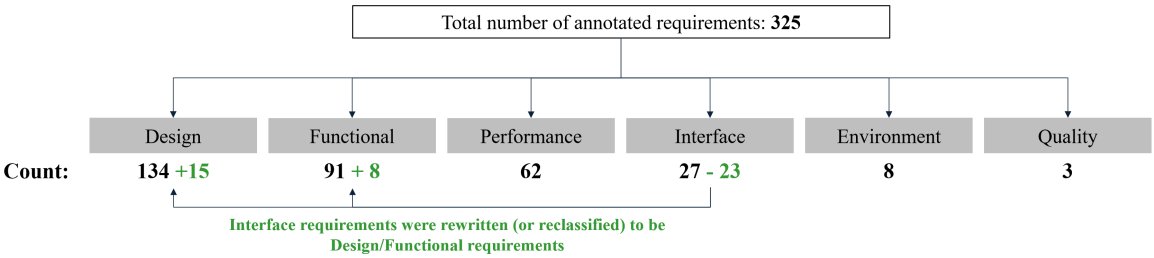


Figure 5.4: Six "types" of requirements were initially considered for the classification corpus. Due to the lack of sufficient examples for Interface, Environment, and Quality requirements, these classes were dropped at a later phase. However, some of the Interface requirements (23) were rewritten (or reclassified) to convert them into either Design or Functional requirements to keep them in the final corpus, which only contains Design, Functional, and Performance requirements.

requirements.

To obtain a more balanced dataset, Environment, and Quality requirements were dropped completely. However, some of the Interface requirements (23) were rewritten (or reclassified) as Design and Functional requirements, as shown in Table 5.8. The rationale for this reclassification was that it is possible to treat the interface as a thing being specified rather than as a special requirement type between two systems.

Table 5.8: Examples showing the modification of Interface requirements into other "types" of requirements

Original **Interface** Requirement	Modified Requirement "type"/category
Each flight recorder shall be supplied with airspeed data.	The airplane shall supply the flight recorder with airspeed data. [**Functional Requirement**]
Each flight recorder shall be supplied with directional data.	The airplane shall supply the flight recorder with directional data. [**Functional Requirement**]
The state estimates supplied to the flight recorder shall meet the aircraft-level system requirements and the functionality specified in Section 23-2500.	The state estimates supplied to the flight recorder shall meet the aircraft level system requirements and the functionality specified in Section 23-2500. [**Design Requirement**]

The final classification dataset includes 149 Design requirements, 99 Functional requirements, and 62 Performance requirements (see Figure 5.4). Lastly, the labels attached to the requirements (design requirement, functional requirement, and performance requirement) were converted into numeric values: 0, 1, and 2, respectively.

The final form of the dataset is shown in Table 5.9.

Table 5.9: Requirement classification dataset format (0: Design; 1: Functional; 2: Performance)

Requirements	Label
Each cockpit voice recorder shall record voice communications transmitted from or received in the airplane by radio.	1
Each recorder container must be either bright orange or bright yellow.	0
Single-engine airplanes, not certified for aerobatics, must not have a tendency to inadvertently depart controlled flight.	2
Each part of the airplane must have adequate provisions for ventilation and drainage.	0
Each baggage and cargo compartment must have a means to prevent the contents of the compartment from becoming a hazard by impacting occupants or shifting.	1

The requirements dataset was collected and annotated by a single human annotator with expertise in the aerospace domain, which took a total of two months. An SME was consulted during the process and various iterations of labeling and review were conducted to ensure the consistency of the labeling.

After having developed two different aerospace corpora, the next step was to use them to fine-tune BERT LM for aerospace NER and requirements classification. Various variants of BERT were fine-tuned for aerospace NER and requirements classification, the detailed methodology for which is discussed in the next sections.

5.2 Fine-tuning BERT for aerospace NER (aeroBERT-NER)

The detailed methodology for fine-tuning BERT for aerospace named-entity recognition (NER) is discussed below.

5.2.1 Preparing the dataset for fine-tuning BERT for aerospace NER

BERT LM expects the input sequence to be in a certain format. The 1432 sentences in the aerospace corpus were divided into training (90%) and validation sets (10%).

The text in the training set was then tokenized using the WordPiece tokenizer. If a word is not present in BERT's vocabulary, the WordPiece tokenizer splits it into subwords. "##" is used as a prefix to denote that the previous string is not whitespace (Figure C.7). Therefore, tokens with "##" as a prefix should be concatenated with the preceding token when converting to a string. It is important to use the same tokenizer that the model was pre-trained with.

The maximum length of the input sequence is decided upon and was set to 175 after examining the distribution of lengths of all sequences in the training set (Figure 5.5). The length of the longest sequence in the aerospace corpus was found to be 172, while the 95th percentile was found to be 68. If any sequence has a length less than the set maximum length, it is post-padded with [PAD] tokens till the sequence length is equal to the maximum length. The sequences which are longer than 175, were truncated. It is crucial to consider the lengths of sequences when working with language models since they have limitations in processing text of certain lengths. If a sequence exceeds the maximum length the language model can handle, the excess text is truncated, resulting in the loss of valuable information.

Special tokens such as [CLS], [SEP], and [PAD] were added to every requirement. Post-padding was performed. All the tokens were converted into their respective "ids" (numbers), and all the [PAD] tokens were tagged as 0. This gives the model an idea about which tokens carry "actual" information as compared to padding tokens. In addition, the tags associated with each sequence were converted into "ids" as well, as shown in Table 5.10. Lastly, attention masks (1 for "real" tokens, and 0 for [PAD] tokens) were obtained for all the sequences in the training set.

Table 5.11 shows an example of a sequence, its associated ids, tags, and attention mask for the sentence, "*It must be shown by analysis or test, or both, that each operable reverser can be restored to the forward thrust position.*" Three columns serve as an input to the BERT model for fine-tuning, namely *token ID*, *tag ID*, and *attention*

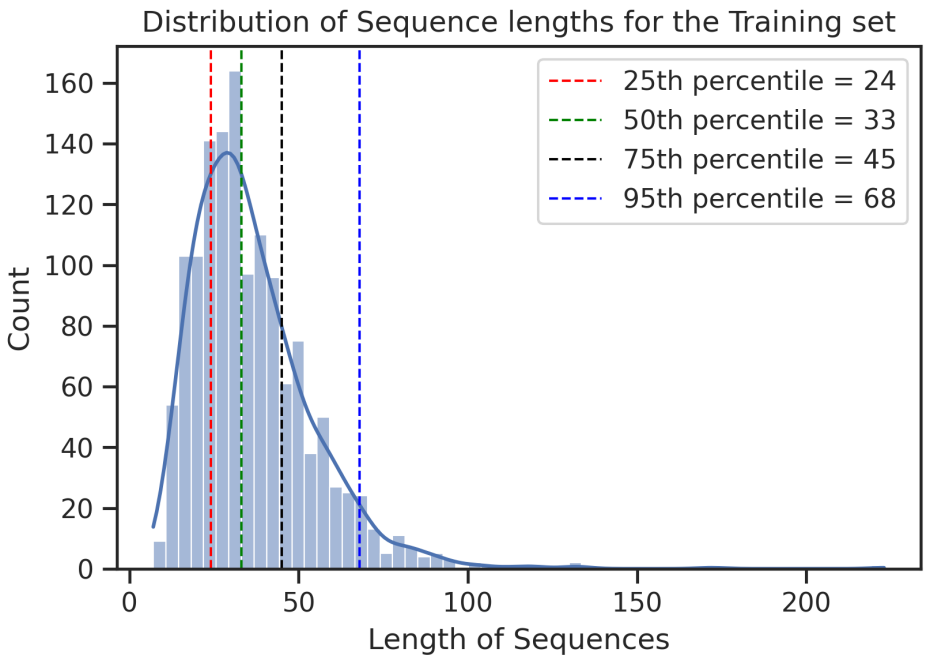


Figure 5.5: Choosing maximum length of the input sequence for training aeroBERT-NER

mask.

5.2.2 Fine-tuning BERT for aerospace NER

The text corpus with annotated named entities was used for the fine-tuning process during which the BERT LM parameters were fine-tuned for NER within the aerospace domain. All four variants of BERT were fine-tuned, namely BERT_{LU}, BERT_{LC}, BERT_{BU}, and BERT_{BC} to obtain their aeroBERT-NER counterparts. A full pass over the training set was performed in each epoch. The batch size was set to 32. Adam optimizer with a learning rate of 3×10^{-5} was used and the model was trained for 20 epochs. The dropout rate was set to the default value of `0.1` to promote the generalizability of the model and avoid overfitting. The training losses were tracked and the norms of the gradients were clipped to avoid the "exploding gradient" problem. The model performance was measured on the validation set for each epoch. The performance metrics precision, recall, and F1 score were used to

Table 5.10: NER tags and their corresponding IDs that were used for this work

NER Tag	Tag ID
I-VAL	0
I-SYS	1
I-RES	2
B-ORG	3
I-ORG	4
B-SYS	5
O	6
B-RES	7
B-VAL	8
B-DATETIME	9
I-DATETIME	10
PAD	11

evaluate the model on the validation set since the dataset was imbalanced in regard to different named entity classes. Model training/validation was carried out many times to make sure that the results were robust and reliable. The fine-tuning process took 268.6 seconds for aeroBERT-NER$_{\text{BC}}$ on an NVIDIA Quadro RTX 8000 GPU with 40 GB VRAM. This training time, however, more than doubles to 873.9 seconds for aeroBERT-NER$_{\text{LC}}$ on the same GPU. This difference in training times can be attributed to the fine-tuning of 110M parameters for aeroBERT-NER$_{\text{BC}}$ as compared to 340M for aeroBERT-NER$_{\text{LC}}$.

Figure 5.6 shows the methodology used for fine-tuning BERT for NER for the requirement, *"Trim control systems must be designed to prevent creeping in flight"*. The embeddings for the WordPiece tokens are fed into the pre-trained LM to compute the token representations by the BERT encoder. The representations are then fed through a linear classification layer. Here, scores measuring how likely the token belongs to each named entity category are computed. The token is classified into the category with the highest score. For example, *Trim* is the beginning of a system name (B-SYS) in this case. This process is repeated for every token in the sequence to classify it into a particular NER category.

To explore the trends in F1 scores on the validation set, several subsets of the

Table 5.11: Tokens, token ids, tags, tag ids, and attention masks

WordPiece Token	Token ID	Tag	Tag ID	Attention Mask
it	1122	O	6	1
must	1538	O	6	1
be	1129	O	6	1
shown	2602	O	6	1
by	1118	O	6	1
analysis	3622	O	6	1
or	1137	O	6	1
test	2774	O	6	1
,	117	O	6	1
or	1137	O	6	1
both	1241	O	6	1
,	117	O	6	1
that	1115	O	6	1
each	1296	O	6	1
opera	4677	O	6	1
##ble	2165	O	6	1
reverse	7936	B-SYS	5	1
##r	1197	I-SYS	5	1
can	1169	O	6	1
be	1129	O	6	1
restored	5219	O	6	1
to	1106	O	6	1
the	1103	O	6	1
forward	1977	O	6	1
thrust	7113	O	6	1
position	1700	O	6	1
.	119	O	6	1
[PAD]	0	O	11	0
...	...	...	...	...
[PAD]	0	O	11	0

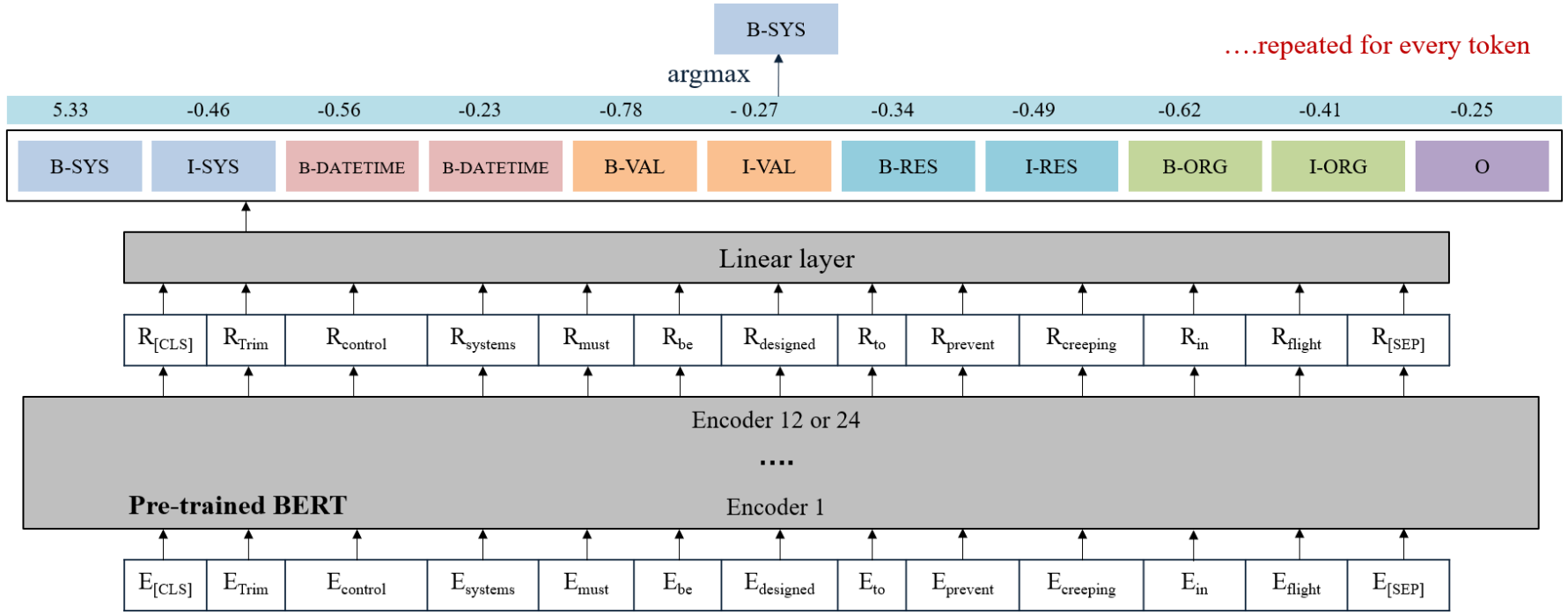


Figure 5.6: The detailed methodology used for full-fine-tuning of BERT LM for aerospace NER is shown here. $E_{[name]}$ represents the embedding for that particular WordPiece token which is a combination of position, segment, and token embeddings. $R_{[name]}$ is the representation for every token after it goes through the BERT model. This representation then passes through a linear layer and is classified into one of the NER categories by the highest estimated probability.

NER on aerospace text.

5.3 Fine-tuning BERT for aerospace requirements classification (aeroBERT-Classifier)

The detailed methodology for fine-tuning BERT for aerospace requirements classification is discussed below.

5.3.1 Preparing the dataset for fine-tuning BERT for classification of aerospace requirements

As mentioned previously, LMs expect inputs in a certain format, and this may vary from one LM to another based on how the model was pre-trained. The dataset was split into training (90%) and test (10%) sets containing 279 and 31 samples, respectively (the corpus contains a total of 310 requirements). Table 5.12 provides a detailed breakdown of the count of each type of requirement in the training and test sets. The LM was fine-tuned on the training set, whereas the model performance was tested on the test set, which the model had not been exposed to during training.

Table 5.12: Breakdown of the types of requirements in the training and test set for aeroBERT-Classifier

Requirement type	Training set count	Test set count
Design (0)	136	13
Functional (1)	89	10
Performance (2)	54	8
Total	**279**	**31**

The text in the training set was then tokenized using the WordPiece tokenizer. The maximum length for the input sequences was set to 100 after examining the distribution of lengths of all sequences (requirements) in the training set (Figure 5.7). All the sequences with a length less than the set maximum length were post-padded

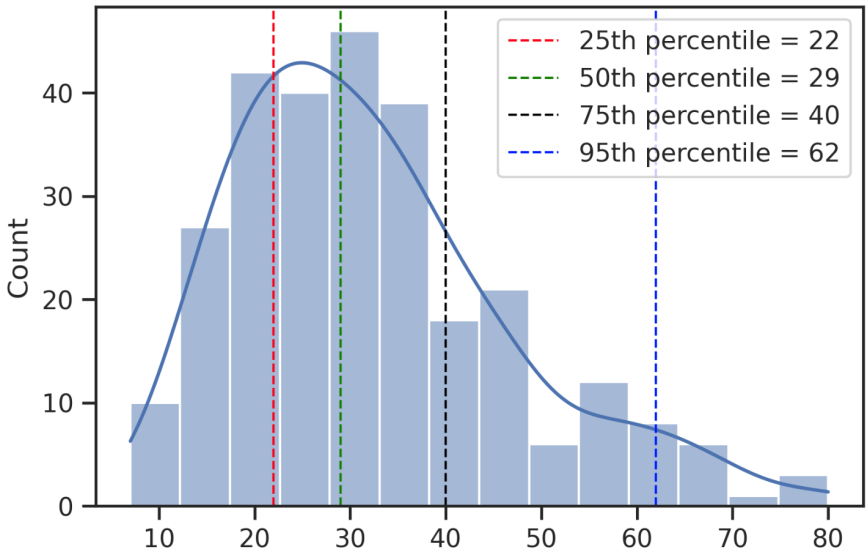


Figure 5.7: Figure showing the distribution of sequence lengths in the training set used for training aeroBERT-Classifier. The 95^{th} percentile was found to be 62. The maximum sequence length was set to 100 for the aeroBERT-Classifier model.

with [PAD] tokens till the sequence length was equal to the maximum length. The sequences which are longer than 100, were truncated. Special tokens such as [CLS], [SEP], and [PAD] were added to every requirement.

All four variants of BERT were fine-tuned, namely BERT_{LU}, BERT_{LC}, BERT_{BU}, and BERT_{BC} to obtain their aeroBERT-Classifier counterparts. Hence, the input sequences were either *cased* or *uncased* depending on the BERT variant being fine-tuned. The tokens present in every sentence were mapped to their respective IDs in BERT's vocabulary of 30,000 tokens. Lastly, attention masks (1 for "real" tokens, and 0 for [PAD] tokens) were obtained for all the sequences in the training set. Only the *token ID* and the *attention mask* columns are used for fine-tuning BERT for requirements classification, as shown in Table 5.13. In addition to these columns, the model is provided with a label for each requirement example.

Table 5.13: Tokens, token IDs, and attention masks for the requirement "*It must be shown by analysis or test, or both, that each operable reverser can be restored to the forward thrust position*" is shown. Only token IDs and attention masks along with the requirement label are provided as inputs to the BERT model for fine-tuning.

WordPiece Token	Token ID	Attention Mask
[CLS]	101	1
it	1122	1
must	1538	1
be	1129	1
shown	2602	1
by	1118	1
analysis	3622	1
or	1137	1
test	2774	1
,	117	1
or	1137	1
both	1241	1
,	117	1
that	1115	1
each	1296	1
opera	4677	1
##ble	2165	1
reverse	7936	1
##r	1197	1
can	1169	1
be	1129	1
restored	5219	1
to	1106	1
the	1103	1
forward	1977	1
thrust	7113	1
position	1700	1
.	119	1
[SEP]	102	1
...	...	...
[PAD]	0	0

5.3.2 Fine-Tuning BERT LM for aerospace requirements classification

A pre-trained BERT model with a linear classification layer on the top is loaded from the *Transformers* library from HuggingFace (`BertForSequenceClassification`). This model and the untrained linear classification layer (full-fine-tuning) are trained on the classification corpus created previously (Table 5.9).

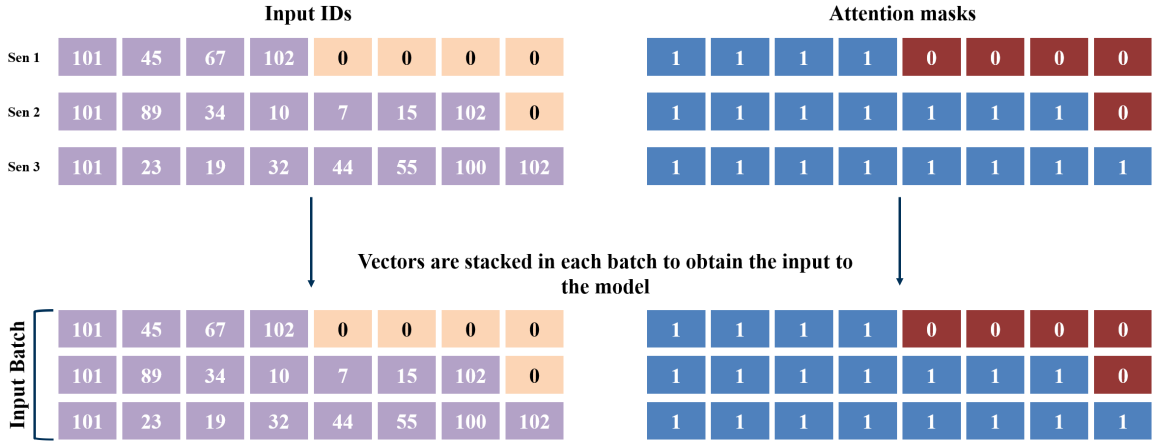


Figure 5.8: The example demonstrates a classification dataset with a batch size of three sentences, along with the input IDs for the corresponding tokens and their respective attention masks. However, it's important to note that the batch size for the aeroBERT-Classifier is actually 16. The special tokens [CLS], [PAD], and [SEP] are assigned the IDs 101, 0, and 102 respectively.

The batch size was set to 16 (Figure 5.8) and the model was trained for 20 epochs. The model was supplied with three tensors for training: 1) input IDs; 2) attention masks; and 3) labels for each example. The `AdamW` optimizer [124] with a learning rate of 2×10^{-5} was used. The previously calculated gradients were cleared before performing each backward pass. In addition, the norm of gradients was clipped to `1.0` to prevent the "exploding gradient" problem. The dropout rate was set to the default value of `0.1` (after experimenting with other rates) to promote the generalizability of the model and speed up the training process. The model was trained to minimize the cross-entropy loss function.

The model performance on the test set was measured by calculating metrics,

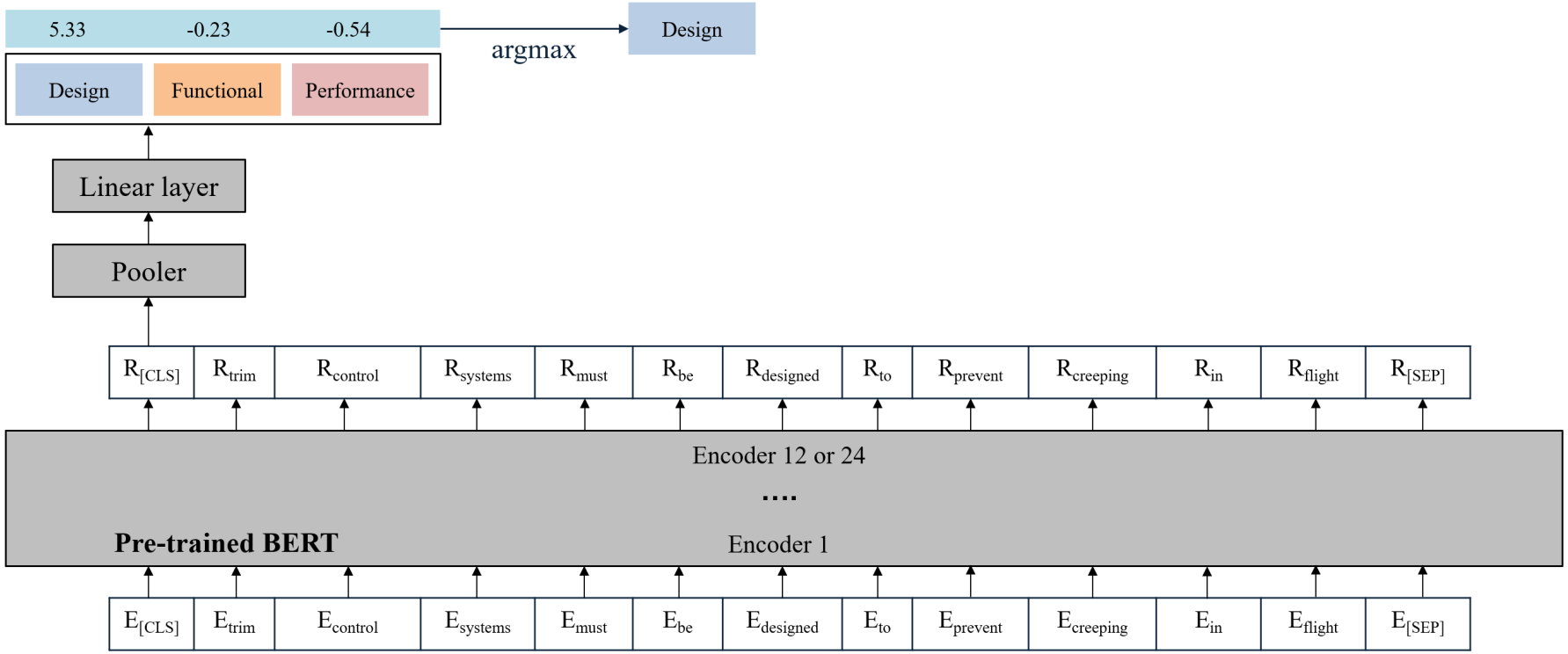


Figure 5.9: The detailed methodology used for full-fine-tuning of BERT LM for aerospace requirements classification is shown here. E_{name} represents the embedding for that particular WordPiece token which is a combination of position, segment, and token embeddings. R_{name} is the representation for every token after it goes through the BERT model. Only $R_{[CLS]}$ is used for requirement classification since its hidden state contains the aggregate sequence representation.

including the F1 score, precision, and recall. Model training and testing were carried out multiple times to make sure that the model was robust and reliable. The fine-tuning process took only 39 seconds for aeroBERT-Classifier$_{\mathrm{BU}}$ and 174 seconds for aeroBERT-Classifier$_{\mathrm{LU}}$ on an NVIDIA Quadro RTX 8000 GPU with a 40 GB VRAM. The short training time can be attributed to the small training set.

Figure 5.9 shows a rough schematic of the methodology used for fine-tuning the BERT LM for requirement classification for the requirement, *"Trim control systems must be designed to prevent creeping in flight"*. The token embeddings are fed into the pre-trained LM and the representations for these tokens are obtained after passing through 12 encoder layers. The representation for the first token ($\mathrm{R}_{[CLS]}$) contains the aggregate sequence representation and is passed through a pooler layer (with a Tanh activation function) and then a linear classification layer. Class probabilities for the requirement belonging to the three categories (design, functional, and performance) are estimated and the requirement is classified into the category with the highest estimated probability, *'Design'* in this case.

In order to evaluate its overall performance, the various variants of aeroBERT-Classifier were compared with other language models that employ different architectures. Details of the models used in the comparison are presented in Table 5.14.

While two of the models employed for comparison with aeroBERT-Classifier are based on transformers (Table 5.14), the Bi-LSTM (with GloVe) utilizes pre-trained GloVe embeddings to train the Bi-LSTM from scratch. By leveraging the knowledge embedded in the pre-trained word embeddings, which have been trained on a large corpus of text data, the model initialization process accelerates the training process and boosts the performance of the model, particularly when training data is inadequate.

Table 5.14: To evaluate the performance of aeroBERT-Classifier, several language models with distinct architectures and training methods were employed for benchmarking purposes.

Model	Model Architecture	Type of Training
GPT-2	Transformer-based model that contains only decoder blocks	Fine-tuned on aerospace classification corpus
Bi-LSTM (with GloVe)	To capture context, pre-trained word embeddings are utilized in conjunction with a Bi-directional long-short term memory (Bi-LSTM) model	Trained from scratch on aerospace classification corpus
bart-large-mnli	Uses a transformer-based architecture to process two input sentences by encoding them, followed by the computation of a similarity score between the encoded representations of the sentences. This similarity score is then utilized to determine the relationship between the two sentences, which could either be entailment, contradiction, or neutral.	Zero-shot classification (ZSL) and is not trained on aerospace classification corpus

5.4 Fine-tuning BERT for POS tagging of aerospace text

The initial proposition was to fine-tune BERT LM for POS tagging of aerospace text. Upon further inspection, however, an off-the-shelf LM called `flair/pos-english` [114] was deemed adequate for POS tagging aerospace text. This LM is trained on Ontonotes [125] which contains text from news articles and broadcasts, telephone conversations, etc. Despite the differences in the training data used for training `flair/pos-english` and aerospace text, this LM is expected to work since POS tags lie at the very basis of the English language and hence are expected to generalize fairly well to previously unseen text.

Another reason for using an off-the-shelf LM for POS tagging is training/fine-tuning a POS model requires a large amount of annotated text given the amount of

intraclass variations for each POS class. Hence, training such a model from scratch would require a substantial amount of annotated aerospace POS datasets as compared to the small annotated corpora used for aeroBERT-NER and aeroBERT-Classifier.

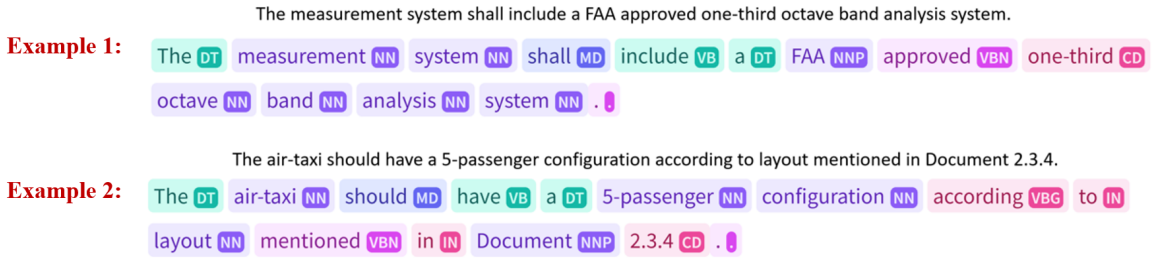


Figure 5.10: Demonstration of intraclass variations in POS tags

Figure 5.10 shows two examples illustrating intraclass variations seen in POS tags using `flair/pos-english` LM [114]. In example 1, "measurement", "system", etc. are tagged as Nouns (NN). However, in example 2, "5-passenger" was also tagged as a Noun (NN). Despite both being nouns, the word "5-passenger" looks very different as compared to the other two. Similarly, "one-third" (from example 1) and "2.3.4" (from example 2) are both tagged as Cardinal numbers (CD), despite the variation observed in them. Hence, a copious amount of annotated text data is required for training/fine-tuning a POS (or text chunking model) to capture the intraclass variations. The examples discussed here give a preview of the variations and are not all-encompassing.

The main idea behind using POS tags (and NER) is to be able to identify linguistic patterns (if any) in different types of aerospace requirements. Hence, aeroBERT-Classifier [126] was first used to classify requirements into various types before performing any POS tag or NER-based analysis. Tokens in the three types of requirements (design, function, and performance) were tagged with their respective POS and NEs and then analyzed to observe any patterns that might emerge. These patterns will then be used for the standardization of requirements.

5.4.1 Observations regarding POS tags and patterns in requirements

Figure 5.11 shows a Sankey diagram representing the observed POS tag patterns in all of the requirements that were classified as design requirements by aeroBERT-Classifier. As can be seen from the figure, most of the requirements start with a determiner (DET), namely, *each*, *the*, etc. DET are mostly followed by nouns which can be system names, or part of a system name in case it spans more than one word.

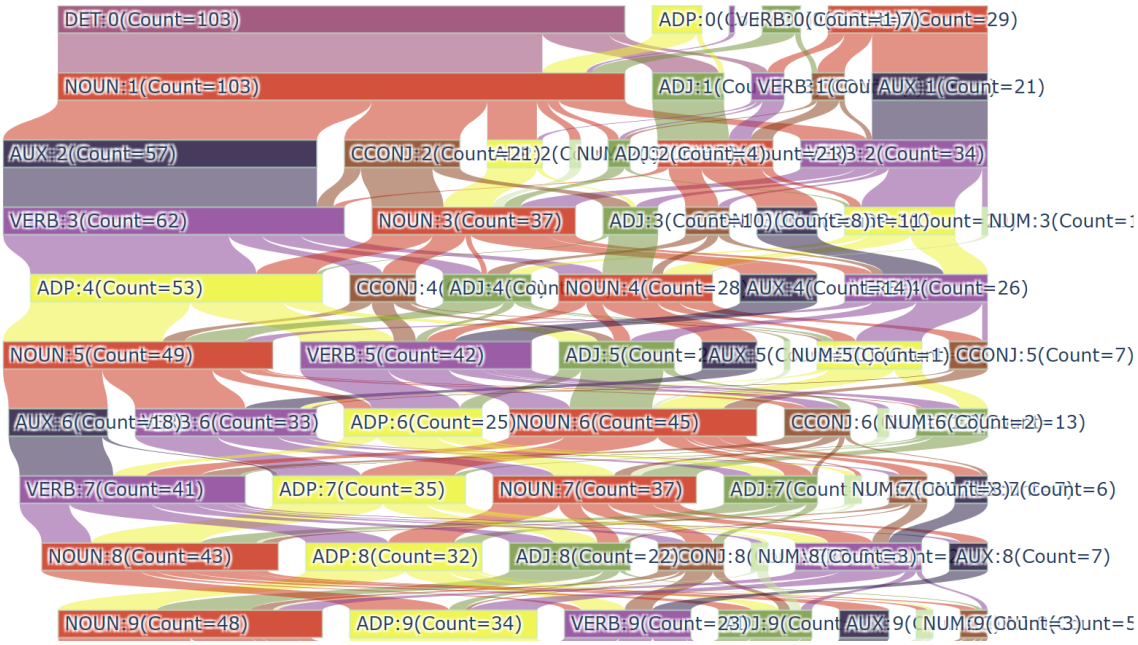


Figure 5.11: Sankey diagram showing the POS tag patterns in design requirements. It is difficult to observe any discernable patterns due to the noise in the POS tags. A part of the figure is shown due to space constraints, however, the full diagram can be found here.

Similarly, Figure 5.12 shows a Sankey diagram representing the observed POS tag patterns in all of the requirements that were classified as functional requirements by the aeroBERT-Classifier. Patterns similar to that observed for design requirements were observed for functional requirements as well.

Lastly, Figure 5.13 shows the POS patterns for performance requirements.

As can be observed from Figure 5.11, Figure 5.12, and Figure 5.13, POS tags by themselves can be noisy (since each word has a POS tag associated with it),

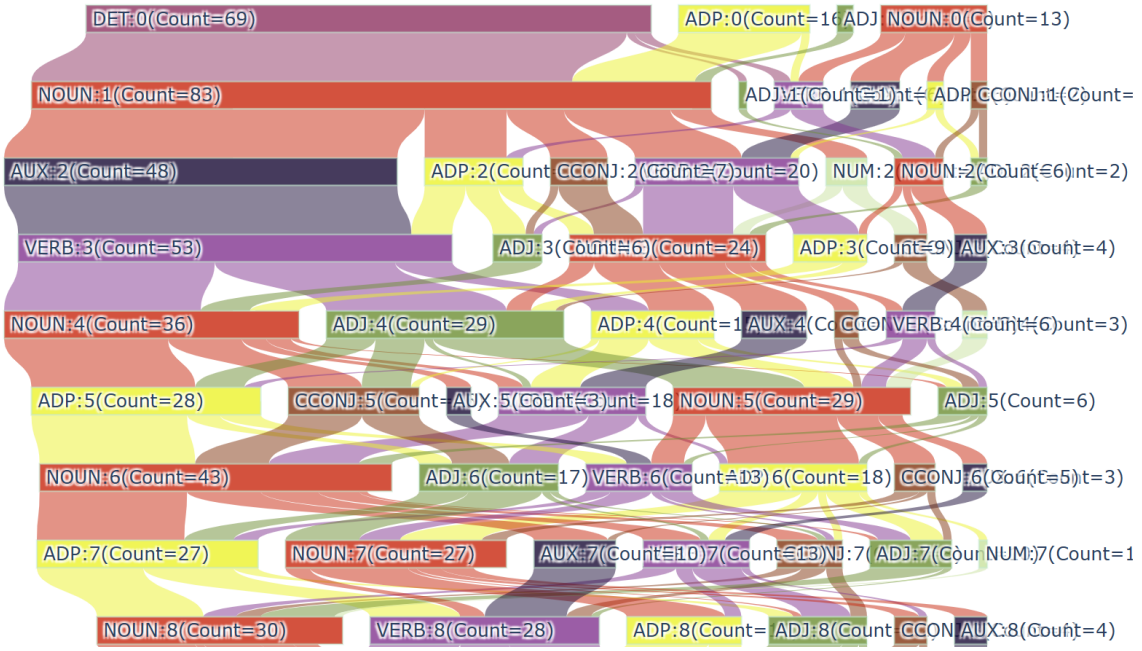


Figure 5.12: Sankey diagram showing the POS tag patterns in functional requirements. Most of the requirements start with a determiner (DET), followed by a NOUN which is usually a system name. A part of the figure is shown due to space constraints, however, the full diagram can be found here.

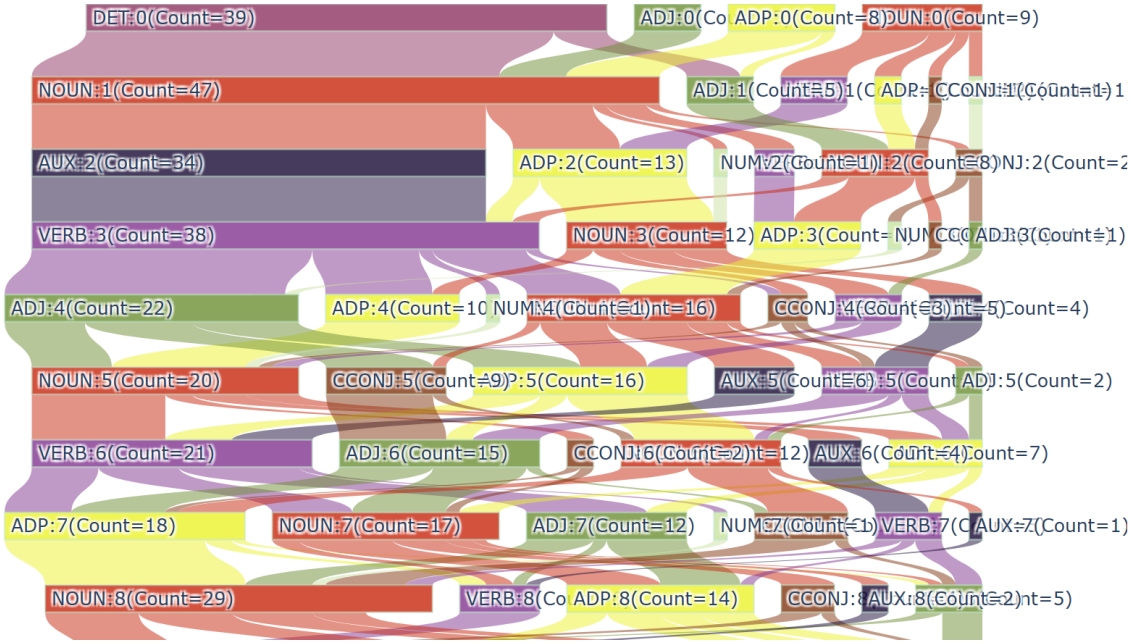


Figure 5.13: Sankey diagram showing the POS tag patterns in performance requirements. A part of the figure is shown due to space constraints, however, the full diagram can be found here.

making them less useful when it comes to identifying patterns in requirements for standardization. For example, in the first requirement in Figure 5.10, the system name (*measurement system* which is a noun) spans two words as compared to the system name in the second requirement (*air-taxi*) which is just one word. The differences in the spans of different words (such as system names, resource names, etc.) and hence their POS tags, render pattern-matching exercises difficult. While, the example presented here demonstrates this in the case of nouns, the same issue exists for other POS tags as well. Therefore, sentence chunks (Figure 5.14), which are an aggregation of POS tags [54], were deemed more useful when it comes to identifying patterns in NL requirements and reorganizing requirement structures to make them follow a standardized template. Sentence/text chunks provide information regarding the syntactic structure of a sentence while reducing the variability as seen in the case of POS tags, hence rendering pattern matching easier.

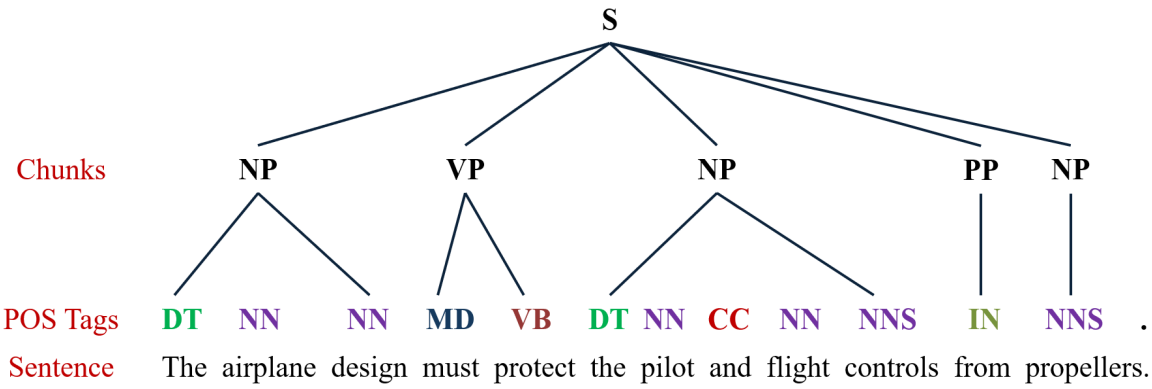


Figure 5.14: An aerospace requirement along with its POS tags and sentence chunks. Each word has a POS tag associated with it which can then be combined together to obtain a higher-level representation called sentence chunks (NP: Noun Phrase; VP: Verb Phrase; PP: Prepositional Phrase).

Various LMs exist for sentence/text chunking [114], and can be accessed via the Hugging Face platform (`flair/chunk-english`). Since NL aerospace requirements are in English, these models are expected to be able to extract the appropriate text chunks and hence do not require fine-tuning an LM on aerospace text. Some of the

Table 5.15: A subset of text chunks along with definitions and examples is shown here [114], [127]. The **bold** text highlights the type of text chunk of interest.

Sentence Chunk	Definition
Noun Phrase (NP)	Consists of a noun and other words modifying the noun (determinants, adjectives, etc.); Example: **The airplane design** must protect **the pilot and flight controls** from **propellers**.
Verb Phrase (VP)	Consists of a verb and other words modifying the verb (adverbs, auxiliary verbs, prepositional phrases, etc.); Example: The airplane design **must protect** the pilot and flight controls from propellers.
Subordinate Clause (SBAR)	Provides more context to the main clause and is usually introduced by subordinating conjunction (because, if, after, as, etc.); Example: There must be a means to extinguish any fire in the cabin **such that** the pilot, while seated, can easily access the fire extinguishing means.
Adverbial Clause (ADVP)	Modifies the main clause in the manner of an adverb and is typically preceded by subordinating conjunction; Example: The airplanes were grounded **until the blizzard stopped**.
Adjective Clause (ADJP)	Modifies a noun phrase and is typically preceded by a relative pronoun (that, which, why, where, when, who, etc.); Example: I can remember the time **when air-taxis didn't exist.**

types of sentence chunks are listed in Table 5.15.

The details regarding the textual patterns identified (using text chunks) in requirements belonging to various types are described below.

Design Requirements

Of the 149 design requirements (Table 5.12), 139 started with a noun phrase (NP), 7 started with a prepositional phrase (PP), 2 started with a subordinate clause (SBAR), and only 1 started with a verb phrase (VP). In 106 of the requirements, NPs were followed by a VP or another NP. The detailed sequence of patterns is shown in

Figure 5.15.

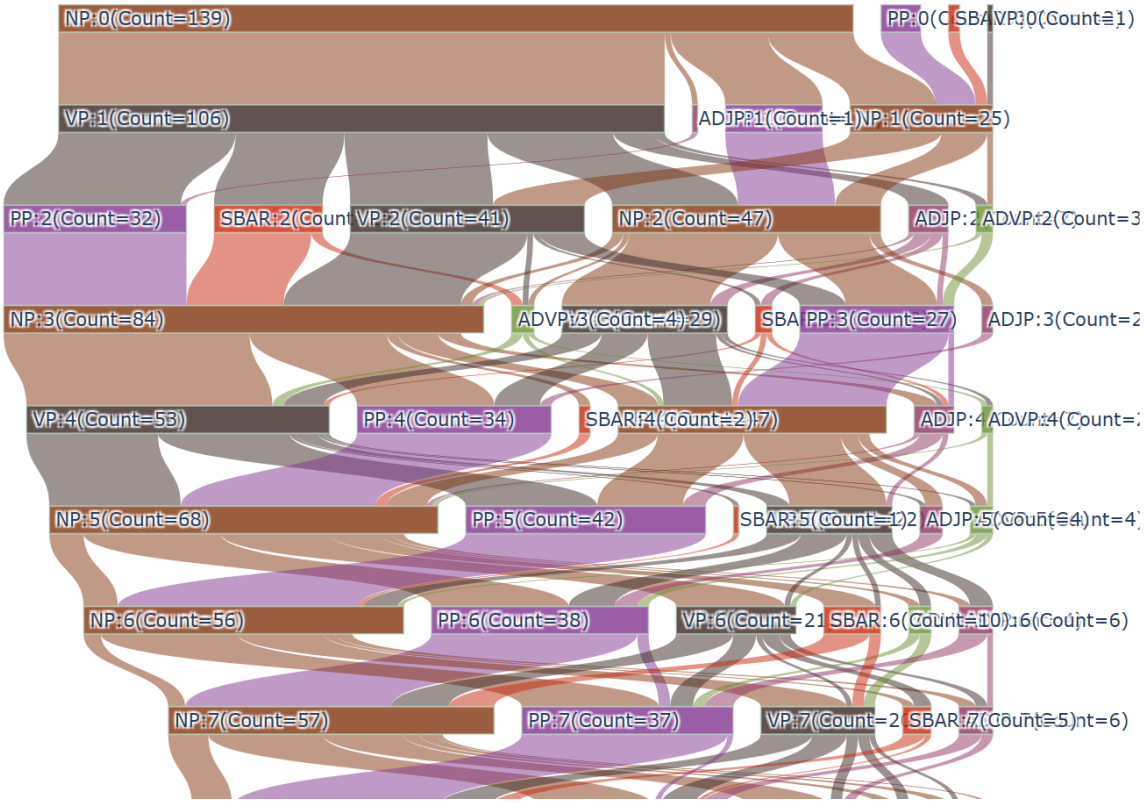


Figure 5.15: Sankey diagram showing the text chunk patterns in design requirements. A part of the figure is shown due to space constraints, however, the full diagram can be found here.

Examples showing design requirements beginning with different types of sentence chunks are shown in Figure 5.16 and Figure 5.17.

Examples 1 through 4 show the design requirements beginning with different types of sentence chunks (PP, SBAR, VP, and NP, respectively) which gives a glimpse into the different ways these requirements can be written and the variation in them. In all cases, the first NP is often the system name and followed by VPs.

It is important to note that in Example 3 (Figure 5.17), the term "*Balancing*" is wrongly classified as a VP. The entire term "*Balancing tabs*" should have been identified as an NP instead. This error can be attributed to the fact that an off-the-shelf sentence chunking model (`flair/chunk-english`) was used and hence it failed to identify "*Balancing tabs*" as a single NP due to the lack of aerospace domain knowl-

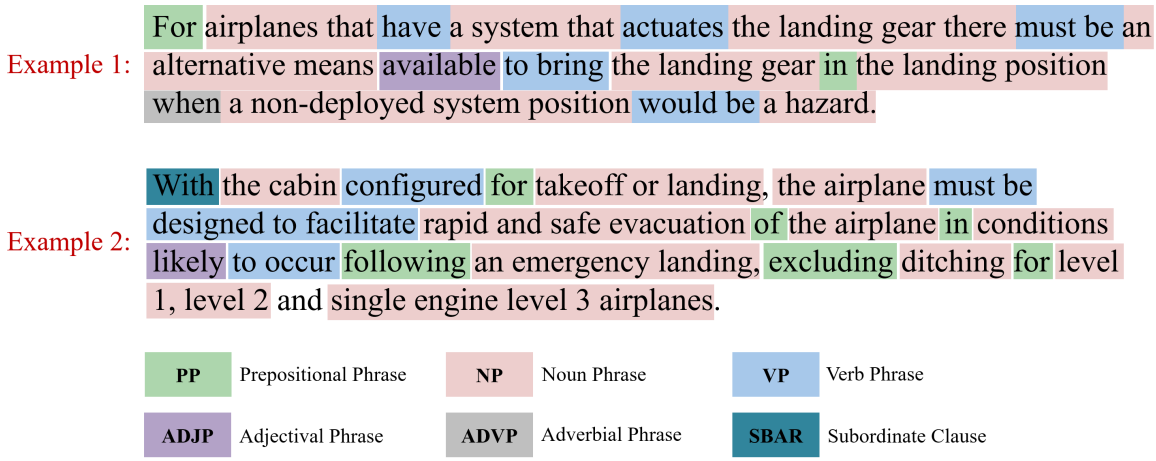


Figure 5.16: Examples 1 and 2 show a design requirement beginning with a prepositional phrase (PP) and subordinate clause (SBAR) which is uncommon in the requirements dataset used for this work. The uncommon starting sentence chunks (PP, SBAR) are however followed by a noun phrase (NP) and verb phrase (VP). Most of the design requirements start with an NP.

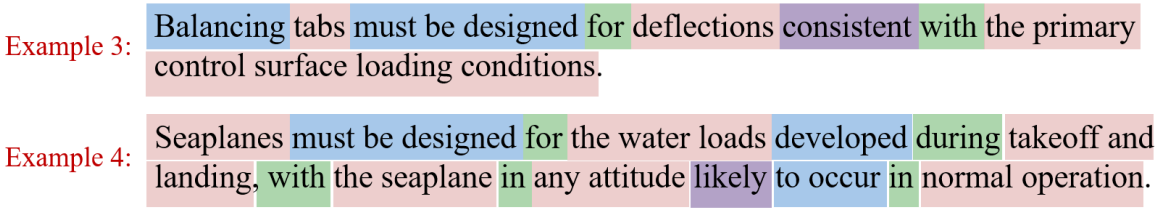


Figure 5.17: Example 3 shows a design requirement starting with a verb phrase (VP). Example 4 shows the requirement starting with a noun phrase (NP) which was the most commonly observed pattern.

edge. Such discrepancies can be resolved by simultaneously accounting for the named entities (NEs) identified by aeroBERT-NER for the same requirement. For this example, "*Balancing tabs*" was identified as a system name (SYS) by aeroBERT-NER which should be a NP by default. In places where the text chunking and NER models disagree, the results from the NER model take precedence since it is fine-tuned on an annotated aerospace corpus and hence has more context regarding the aerospace domain.

Functional Requirements

Of the 100 requirements classified (by aeroBERT-Classifier) as functional, 84 started

with a noun phrase (NP), 10 started with a prepositional phrase (PP), and 6 started with a subordinate clause (SBAR). The majority of the NPs are followed by a VP (69). The detailed sequence of patterns is shown in Figure 5.18.

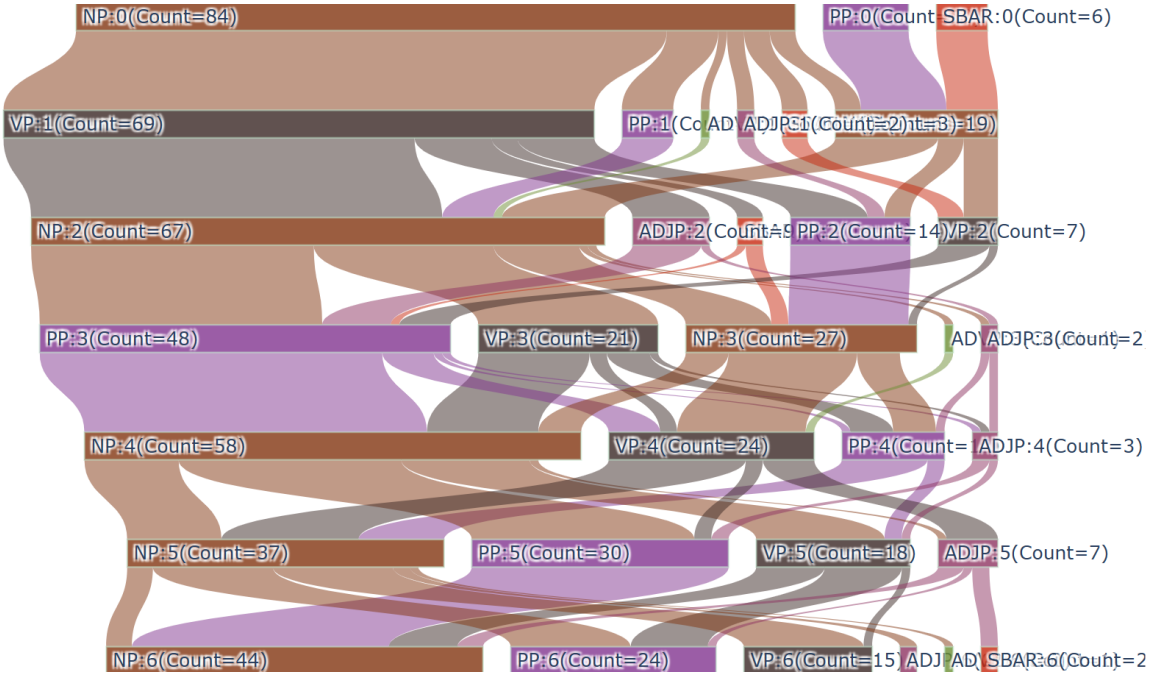


Figure 5.18: Sankey diagram showing the text chunk patterns in functional requirements. A part of the figure is shown due to space constraints, however, the full diagram can be found here.

As can be observed from Figure 5.18, functional requirements used for this work, either start with an NP, PP, or SBAR. Figure 5.19 shows examples of functional requirements beginning with these three types of sentence chunks.

The functional requirements beginning with an NP have system names in the beginning (Example 1 of Figure 5.19). However, this is not the case for requirements that start with a condition, as shown in Example 3.

Performance Requirements

Of the 61 requirements classified as performance, 53 started with a noun phrase (NP), and 8 started with a prepositional phrase (PP). The majority of the NPs are followed by a VP (39). The detailed sequence of patterns is shown in Figure 5.20.

Example 1: Each cockpit voice recorder shall record voice communications transmitted from or received in the airplane by radio.

Example 2: For levels 1,2, and 3, the airplane must maintain lateral and directional trim in cruise without further force upon, or movement of, the primary flight controls or corresponding trim controls by the pilot, or the flight control system.

Example 3: Unless the failure of an automatic power or thrust control system is extremely remote, the system must provide a means for the flightcrew to override the automatic function.

Figure 5.19: Examples 1, 2, and 3 show functional requirements starting with an NP, PP, and SBAR. Most of the functional requirements start with an NP, however.

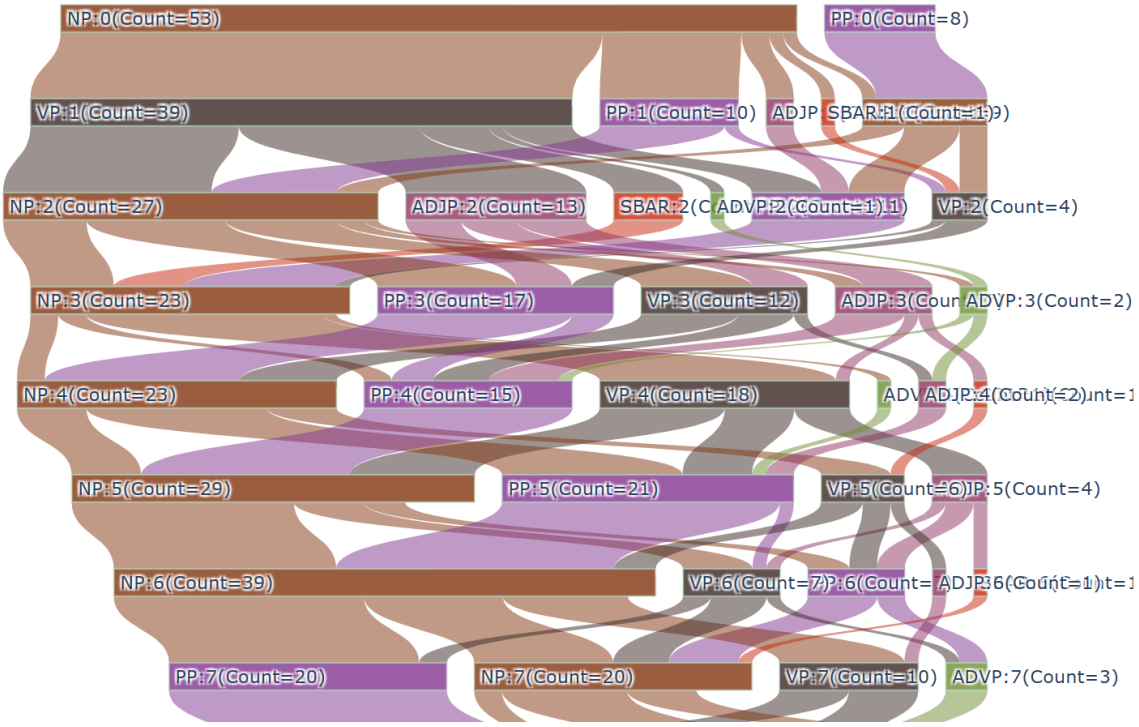


Figure 5.20: Sankey diagram showing the text chunk patterns in performance requirements. A part of the figure is shown due to space constraints, however, the full diagram can be found here.

Examples of performance requirements starting with an NP, and PP are shown in Figure 5.21. The requirement starting with an NP usually starts with a system name, which is in line with the trends seen in the design and functional requirements. Whereas, the requirements starting with a PP usually have a condition in the beginning.

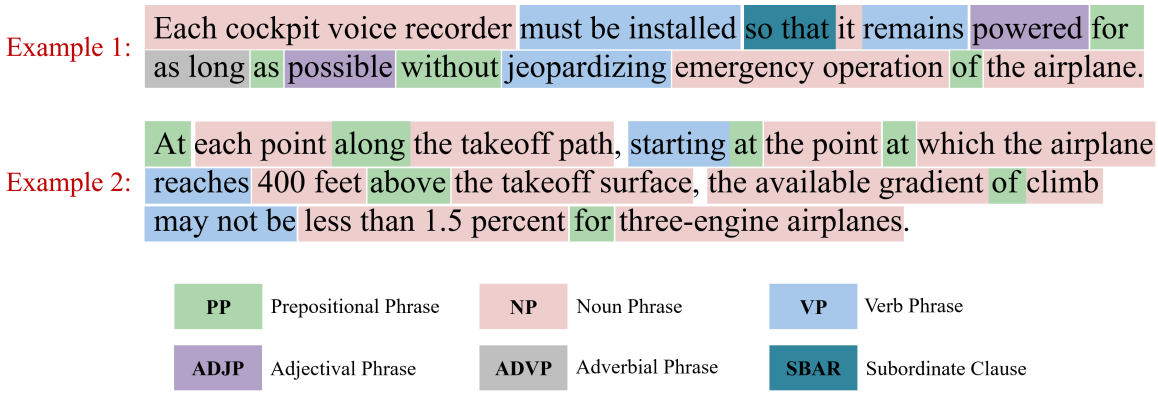


Figure 5.21: Examples 1 and 2 show performance requirements starting with NP and PP respectively.

In example 2 (Figure 5.21), cardinal numbers, such as "400 feet", and "1.5 percent" are both tagged as NP, however, there is no way to distinguish between the different variations in NPs (NPs containing cardinal numbers vs not). Using aeroBERT-NER in conjunction with `flair/chunk-english` is expected to clarify different types of entities beyond their text chunk tags which are helpful for ordering entities in a requirement. The same idea applies to resources (RES, for example, Section 25-395) as well.

5.5 Standardization of requirements

The main goal of this dissertation is to be able to standardize requirements. To that end, two language models (aeroBERT-Classifier and aeroBERT-NER) were developed, and another off-the-shelf model (`flair/chunk-english`) was used for the standardization. While the initial idea of standardization was to develop a methodology for the identification of boilerplates, this changed upon discovering the new capabilities brought forth by the LMs that were developed. Hence, in addition to boilerplate identification, the creation of a requirements table (capable of capturing requirements and system properties in a tabular environment) was deemed feasible. To summarize, the three LMs help with the standardization of requirements in two

ways (Figure 5.22), namely, 1) the creation of a requirements table, and 2) boilerplate identification.

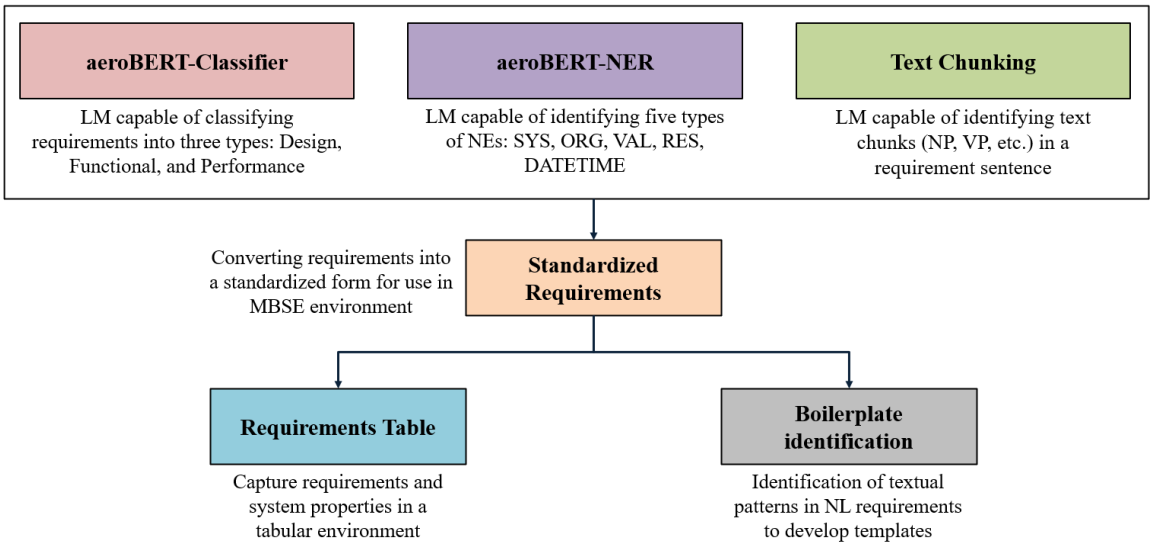


Figure 5.22: Pipeline for converting NL requirements to standardized requirements using various LMs.

Details regarding the background and methodology for the development of the requirements table will be discussed next.

5.5.1 Requirements Table

The subsequent sections furnish an overview of requirements tables and elucidate the approach for constructing a requirements table using language models.

Background

A requirement table can be created in SysML and is used to capture requirements in a spreadsheet-like environment. Each row in the table represents a requirement, and additional columns can be added to capture the attributes assigned to the requirement or system model elements related to it. This table can be used to filter requirements of interest and their associated properties. In addition, given its tabular format, the requirements table can be easily exported into a Microsoft Excel spreadsheet [128]. This table can be used for automated requirements analysis and

modeling as requirements change with time, hence leading to time and cost savings [129]. A typical SysML requirements table is shown in Figure 5.23.

#	Id	Name	Text	Satisfied By
1	S0.0	Original Statement	Describe a system for purifying dirty water. - Heat dirty water and condense steam are performed by a Counter Flow Heat Exchanger - Boil dirty water is performed by a Boiler. Drain residue is performed by a Drain. The water has properties: vol = 1 liter, density 1 gm/cm3, temp 20 deg C, specific heat 1cal/gm deg C, heat of vaporization 540 cal/gm.	
2	S0.1	Elevation	The water distiller shall be able to operate at least 2 meters vertically above the source of dirty water.	
3	S1.0	Purify Water	The system shall purify dirty water.	
4	S2.0	Heat Exchanger	Heat dirty water and condense steam are performed by a Counter Flow Heat Exchanger	Heat Exchanger Heat Exchanger Heat Exchanger Heat Exchanger
5	S3.0	Boiler	Boil dirty water is performed by a Boiler.	Boiler Boiler Boiler Boiler
6	S4.0	Drain	Drain residue is performed by a Drain.	drain : Valve Valve drain : Valve Valve drain : Valve Valve Valve drain : Valve
7	S5.0	Water Properties	Water has properties: density 1 gm/cm3, temp 20 deg C, specific heat 1cal/gm deg C, heat of vaporization 540 cal/gm.	
8	S5.1	Water Initial Temp	Water has an initial temp 20 deg C	

Figure 5.23: A SysML requirements table with four columns, namely, Id, Name, Text, and Satisfied by. It typically contains these four columns by default, however, more columns can be added to capture other properties pertaining to the requirement.

Riesener et al. [130] illustrate a pipeline for the generation of the SysML requirements table using a dictionary-based method. Relevant domain-specific keywords were added to a dictionary, and these words were then extracted when they occurred in mechatronics requirements text. Technical units along with related quantities, material properties, and manufacturing processes were the named-entity types of interest for this work [130]. The amount of effort required to create a dictionary for the extraction of words of interest is tremendous and does not generalize across different projects. In addition, this dictionary needs to be updated from time to time to keep up with the occurrence of new units, etc. Therefore, employing language models to extract pertinent named entities (like system names, values, units, etc.) is more adaptable and scalable than dictionary-based methods, which can either be

unchanging or require manual revisions, rendering them arduous.

Despite the advantages offered by a requirement table, the creation of such a table can be very manual and hence prone to human errors due to repetitive tasks that need to be performed for populating the various columns of interest. Therefore, various LMs can be used to automate the creation of an Excel spreadsheet (requirement table) containing some of the columns of interest which will aid the creation of a similar table in SysML.

Methodology for the creation of requirements table

As discussed previously, the requirement table is helpful for filtering requirements of interest and performing requirements analysis. For the purpose of this work, five columns were chosen to be included in the requirement table, as shown in Table 5.16. The *Name* column was populated by the system name (SYS named entity) identified by aeroBERT-NER [120]. The *Type of Requirement* column was populated after the requirement is classified by aeroBERT-Classifier [126], which is capable of classifying requirements into three types, namely, design, functional, and performance. The *Property* column was populated by all the named entities identified by aeroBERT-NER (except for SYS) in a Python dictionary format where the *named entity type* (RES, VAL, DATETIME, ORG) was set to be dictionary key and the identified named entities are the values (presented in a Python list format). Lastly, the *Related to* column was populated by the system name identified by aeroBERT-NER [120].

The methodology for the identification of textual patterns in aerospace requirements to create boilerplates will be explored in the next subsection.

5.5.2 Requirements Boilerplate

The order of the language models used for identifying requirements boilerplate templates are depicted in Figure 5.24.

Table 5.16: List of language models used to populate the columns of requirement table.

Column Name	Description	Method used to populate
Name	System (SYS named entity) that the requirement pertains to	aeroBERT-NER [120]
Text	Original requirement text	Original requirement text
Type of Requirement	Classification of the requirement as design, functional, or performance	aeroBERT-Classifier [126]
Property	Identified named entities belonging to RES, VAL, DATETIME, and ORG categories present in a requirement related to a particular system (SYS)	aeroBERT-NER [120]
Related to	Identified system named entity (SYS) that the requirement properties are associated with	aeroBERT-NER [120]

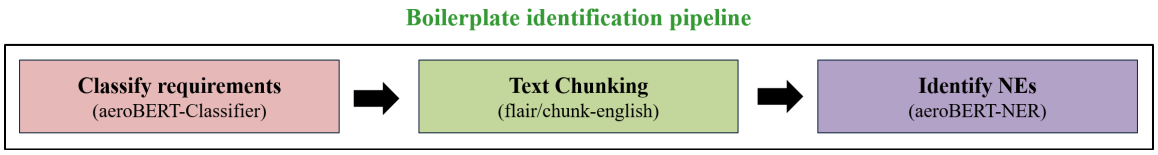


Figure 5.24: Initially, aeroBERT-Classifier is used to classify the requirements. After that, `flair/chunk-english` is employed to chunk the requirement text. Finally, aeroBERT-NER is utilized to detect named entities in the requirements. The results obtained from the three language models are utilized to spot textual patterns in different types of requirements, which are subsequently employed to generate boilerplate templates.

Initially, all the requirements underwent classification into three categories, namely design, functional, and performance, using the aeroBERT-Classifier language model. Out of the total requirements, 149 were classified as design, 100 as functional, and 61 as performance. Additionally, to maintain consistency among the requirements, all variations such as "should have", "must have", etc., were changed to "shall have". Following this, the requirements were tagged for sentence chunks and named entities using `flair/chunk-english` and aeroBERT-NER, respectively. An illustration exemplifying this process is presented in Figure 5.25.

This was followed by going through the original requirement text and identifying broad high-level patterns. After analyzing the three types of requirements it was discovered that there was a general textual pattern irrespective of the type, as shown

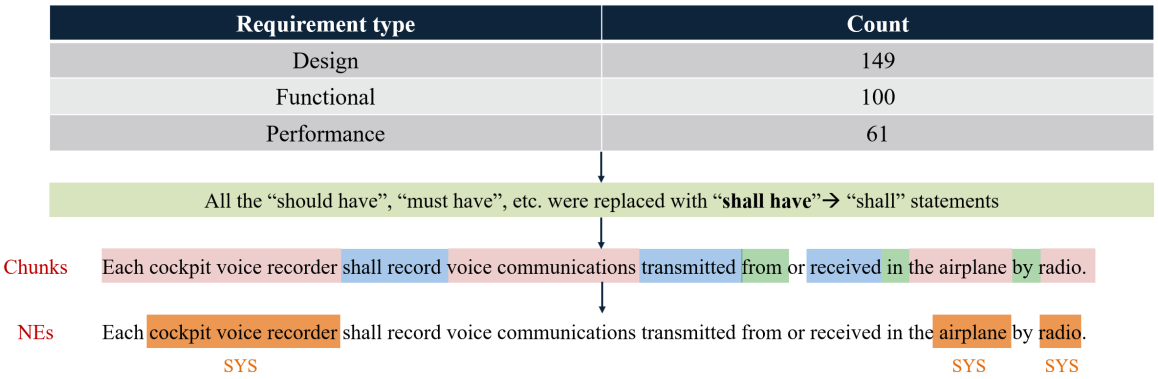


Figure 5.25: Example demonstrating the boilerplate identification pipeline for the requirement "*Each cockpit voice recorder shall record voice communications transmitted from or received in the airplane by radio.*"

in Figure 5.26.

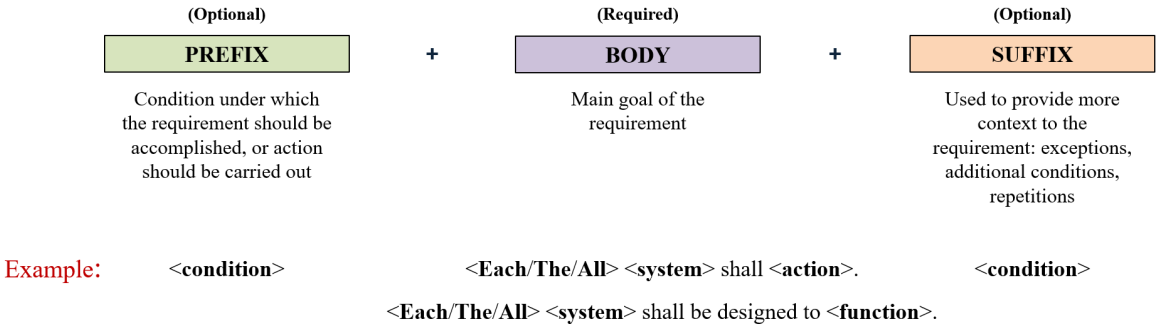


Figure 5.26: The general textual pattern observed in the requirements was ⟨prefix⟩ + ⟨body⟩ + ⟨suffix⟩ out of which *prefix* and *suffix* are optional and can be used to provide more context about the requirement.

The *Body* section of the requirement usually starts with a NP (system name) which (for most cases) contains an SYS named-entity, whereas the beginning of a *Prefix* and *Suffix* is usually marked by a subordinate clause (SBAR) or prepositional phrase (PP), namely, 'so that', 'so as', 'unless', 'while', 'if', 'that', etc. Both the *Prefix* and *Suffix* provide more context into a requirement and thus are likely to be conditions, exceptions, the state of the system after a function is performed, etc. Usually, the *Suffix* can contain various different types of NEs, such as names of resources (RES), values (VAL), other system or sub-system names (SYS) that add more context to the requirement, etc. It is mandatory for a requirement to have a *Body*, however, both

prefixes and suffixes are optional.

The contents of an aerospace requirement can be broadly classified into various elements, such as system, functional attribute, state, condition, etc. The details about the different elements of a requirement are described in Table 5.17 along with examples. The presence or absence of these elements, along with their sequence/order is a distinguishing factor for requirements belonging to different types as well as requirements within a type, hence giving rise to different boilerplate structures.

Table 5.17: Elements of requirement boilerplate templates, their definitions, and examples. The **bold** text highlights the particular element of interest in the example requirement.

Element	Definition
⟨condition⟩	Specifies details about the external circumstance, system configuration, or current system activity under which a system is present while performing a certain function, etc.; Example: **With the cabin configured for takeoff or landing**, the airplane shall have means of egress, that can be readily located and opened from the inside and outside.
⟨system⟩	Name of the system that the requirement is about; Example: All **pressurized airplanes** shall be capable of continued safe flight and landing following a sudden release of cabin pressure.
⟨functional attribute⟩	The function to be performed by a system; Example: The insulation on electrical wire and electrical cable shall be **self-extinguishing**.
⟨state⟩	Describes the physical configuration of a system while performing a certain function; Example: The airplane shall maintain **longitudinal trim** without further force upon, or movement of, the primary flight controls.
⟨design attribute⟩	Provides additional details regarding a system's design; Example: Each recorder container shall have **reflective tape affixed to its external surface** to facilitate its location under water.

Element	Definition
⟨sub-system/system⟩	Specifies any additional system/sub-system that the *main* system shall include, or shall protect in case of a certain operational condition, etc.; Example: Each recorder container shall have **an underwater locating device.**
⟨resource⟩	Specifies any resource (such as another Part of the Title 14 CFRs, a certain paragraph in the same Part, etc.) that the system shall be compliant with; Example: The control system shall be designed for pilot forces applied in the same direction, using individual pilot forces not less than 0.75 times those obtained in accordance with **Section 25-395**.
⟨context⟩	Provides additional details about the requirement; Example: With the cabin configured for takeoff or landing, the airplane shall have means of egress, **that can be readily located and opened from the inside and outside**.
⟨user⟩	Specifies the user of a system, usually a pilot in case flight controls, passengers in case of emergency exits in the cabin, etc.; Example: The airplane design shall protect **the pilot and flight controls** from propellers.
⟨system attribute⟩	Some requirements do not start with a *system name* but rather with certain characteristic of a said system; Example: The **airplane's available gradient of climb** shall not be less than 1.2 percent for two-engine airplane at each point along the takeoff path, starting at the point at which the airplane reaches 400 feet above the takeoff surface.

The information about the sentence chunks and NEs helped in the identification of elements (described in Table 5.17) present in different requirements and hence contributed to the identification of relevant boilerplate patterns. The methodology used for the identification of patterns in requirements is discussed in the following subsection.

5.5.3 Identification of boilerplates by examining patterns in requirements text

This study focuses on developing an agile methodology to identify and create boilerplate templates based on observed patterns in well-written requirements as opposed to proposing generalized boilerplates. Consequently, this initiative is anticipated to aid diverse organizations in devising bespoke boilerplates by examining textual patterns in requirements that are specific to their internal needs. This is beneficial as the structures and types of requirements can differ not only between organizations but also within an organization. For the purpose of this work, certification requirements from Parts 23 and 25 of Title 14 CFRs were used for this work due to their availability to the author.

The requirements are first classified using the aeroBERT-Classifier. The patterns in the sentence chunk and NE tags are then examined for each of the requirements and the identified patterns are used for boilerplate template creation (Figure 5.24, Figure 5.25). Based on these tags, it was observed that a requirement with a ⟨condition⟩ in the beginning usually starts with a prepositional phrase (PP) or subordinate clause (SBAR). Requirements with a condition, in the beginning, were however rare in the dataset used for this work. The initial ⟨condition⟩ is almost always followed by a noun phrase (NP), which contains a ⟨system⟩ name which can be distinctly identified within the NP by using the NE tag (SYS). The initial NP is always succeeded by a verb phrase (VP) which contains the term "shall [variations]". These "[variations]" (shall be designed, shall be protected, shall have, shall be capable of, shall maintain, etc.) were crucial for the identification of different types of boilerplates since they provided information regarding the action that the ⟨system⟩ should be performing (to protect, to maintain, etc.). The VP is followed by a subordinate clause (SBAR) or prepositional phrase (PP) but this is optional. The SBAR/PP is succeeded by a NP or adjective clause (ADJP) and can contain either a ⟨functional attribute⟩, ⟨state⟩, ⟨design attribute⟩, ⟨user⟩, or a ⟨sub-system/system⟩, depending

on the type/sub-type of a requirement. This usually brings an end to the main *Body* of the requirement. The *Suffix* is optional and can contain additional information such as operating and environmental conditions (can contain VAL named entity), resources (RES), and context.

The process of categorizing requirements, chunking, performing Named Entity Recognition (NER), and identifying requirement elements is iterated for all the requirements. The elements are recognized by combining information obtained from the sentence chunker and aeroBERT-NER. The requirements are subsequently classified into groups based on the identified elements to form boilerplate structures. Optional elements are included in the structures to cover any variations that may occur, allowing more requirements to be accommodated under a single boilerplate.

CHAPTER 6
RESULTS AND DISCUSSION

This chapter discusses in detail the results regarding the various corpora and LMs that were developed for the conversion of NL aerospace requirements into semi-machine-readable/standardized requirements. This conversion was facilitated by the development of aeroBERT-NER, aeroBERT-Classifier, and the use of sentence chunking, which will be discussed in the following sections.

6.1 Creation of annotated aerospace corpora

Two different annotated aerospace corpora (or datasets) were created as a part of this dissertation and can be accessed via the `Hugging Face` platform using the Python scripts provided in Appendix A. The details regarding the corpora are listed below:

1. **`archanatikayatray/aeroBERT-NER`**: named entity corpus, which contains aerospace text and requirements that have been annotated for NEs belonging to five different types, namely, SYS, ORG, VAL, RES, and DATETIME

2. **`archanatikayatray/aeroBERT-classification`**: aerospace requirements corpus, which contains requirements that belong to three different types, namely, design, functional, and performance

Due to the technical nature of the data, a single human annotator manually annotated both aforementioned corpora, which took over six months for the annotation process. The annotations were subsequently evaluated by subject matter experts (SMEs) who provided feedback. This feedback led to further revisions being made to ensure consistency across the annotations.

Open-source datasets are scarce when it comes to requirements engineering, especially those specific to aerospace applications. Hence, the above datasets not only enable the present research but are also expected to promote research in the aerospace requirements engineering domain.

6.2 aeroBERT-NER

Four different variants of BERT were fine-tuned to obtain variants of aeroBERT-NER. For example, BERT_{BC} was fine-tuned to obtain aeroBERT-NER_{BC}, and so on. aeroBERT-NER was developed to identify NEs associated with aerospace text, especially requirements. It is capable of identifying five different types of NEs (Table 5.4). This section discusses the performance of aeroBERT-NER on previously unseen aerospace requirements. In addition, a methodology is established to create a glossary using the NEs identified by the LM. Lastly, a comparison between the performance of aeroBERT-NER and $\text{BERT}_{\text{BASE}}$-NER on aerospace requirements is done.

6.2.1 Model performance vs. dataset size

The fine-tuning of different BERT variants was carried out using datasets of different sizes, with the goal to observe the trends in learning and to answer the question, *"how much data is enough for the LM to gain aerospace NER domain knowledge?"*. The F1 scores were considered to be an accurate representation of the model's performance on the validation set since NER is a token classification task. The fine-tuning process was carried out for 20 epochs for each dataset size containing 250, 500, 750, 1000, 1250, and 1423 sentences, respectively. It is important to understand that some of the sentences in the dataset did not contain any named entity. These sentences were included to make sure the model learns that there can be aerospace sentences that do not contain any of the five named entities considered for this work.

Each of these datasets was divided into training (90%) and validation (10%) sets. Figure 6.1 shows the trends in F1 scores for different dataset sizes when the model is trained on the training set and tested on the validation set for 20 epochs.

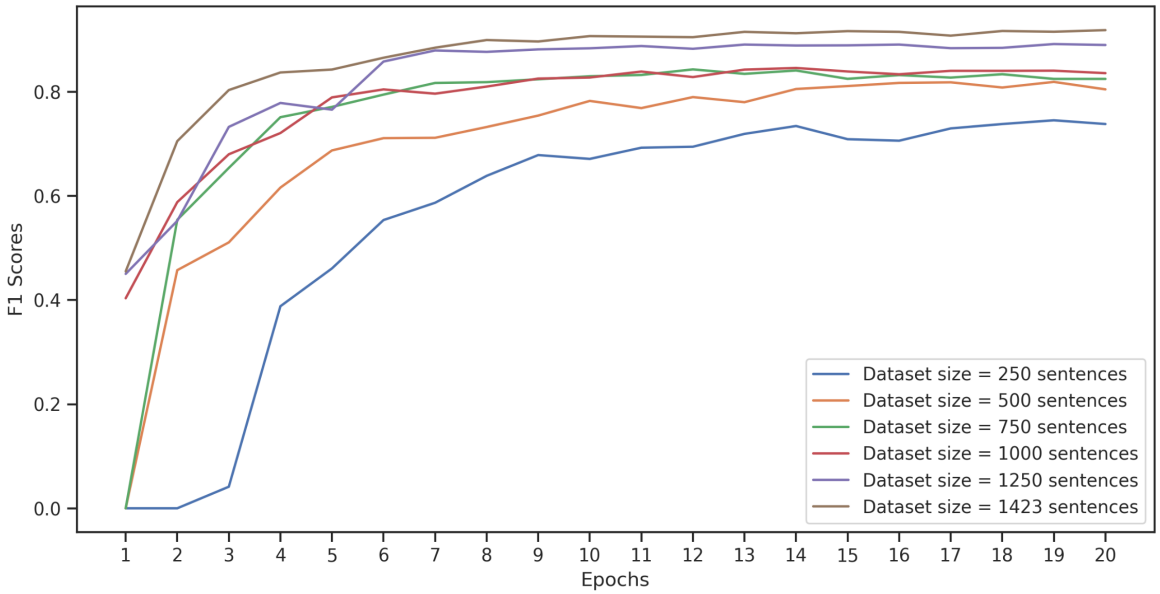


Figure 6.1: Comparing validation performance of aeroBERT-NER when trained and tested on different dataset sizes

The validation F1 score gradually increases as the number of epochs increases, which indicates that the model learns to perform NER effectively. Further, the F1 score increases as the size of the dataset increases. This gain is more noticeable as the dataset increases from 250 to 500 sentences than it is when the dataset increases from 1250 to 1423 sentences. As shown in Table 6.1, enlarging the dataset size does not result in a commensurate increase in the F1 score. This may be due to the fact that certain datasets may have more instances of a particular entity to train on (compared to others), which results in improved overall performance. The BERT LM was able to acquire sufficient aerospace-specific domain knowledge in NER with only 1432 annotated sentences, resulting in a final F1 score of 92%. Since annotating data can be tedious and the addition of more data may only result in a marginal increase in the F1 score, it was determined that no more annotated data would be added.

Table 6.1: Trends observed in F1 scores with the increase in dataset size for training and testing. 90% of the dataset was used for training and 10% was used for testing.

Dataset size (sentences)	Increase in dataset	F1 score	% increase in F1 score
250	-	0.74	-
500	+250	0.80	8.11
750	+250	0.83	3.75
1000	+250	0.84	1.20
1250	+250	0.89	5.95
1432	+182	0.92	3.37

Table 6.2 summarizes the performance of aeroBERT-NER (fine-tuned using 90% of 1423 sentences and run on the validation sets over 20 epochs) as measured by validation precision, recall, and F1 score.

Table 6.2 also includes a weighted average of the metrics across categories to account for the disproportionate number of times a specific entity occurs (e.g. system names occur more frequently than ORG). aeroBERT-NER performs at over 92% in F1 score for aerospace-specific NER, which is a nearly state-of-the-art performance mark in general-purpose NER, where much larger public datasets exist. The state-of-the-art model by [131] achieves an F1 score of 94.6% on the CoNLL-2003 NER dataset [107] which is hand-annotated for four categories (ORG, LOC, PER, and MISC) and is a much larger dataset (Table 3.3).

aeroBERT-NER$_{\text{BC}}$ variant performs the best (both in terms of training time and model performance) as compared to aeroBERT-NER$_{\text{LARGE}}$ variants. In addition both the *cased* variants (aeroBERT-NER$_{\text{LC}}$, and aeroBERT-NER$_{\text{BC}}$) perform better overall. This can be attributed to the fact that preserving *casing* is important in the case of token classification tasks such as NER. To explain further, if the input words are "Georgia Tech", it will be converted to "georgia tech" for a *uncased* LM, whereas the original casing is preserved for a *cased* LM (Figure 6.2).

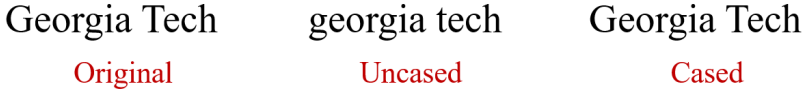


Figure 6.2: Original, cased, and uncased versions for the word "Georgia Tech"

Table 6.2: aeroBERT-NER model performance on the validation set. The highest score for each metric per NER type is shown in **bold**. (P = Precision, R = Recall, F1 = F1 Score)

Models	**DT**			**ORG**			**RES**			**SYS**			**VAL**			**Avg.**
	P	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**F1**
aeroBERT-NER$_{LU}$	0.81	0.88	0.84	0.68	0.67	0.68	0.96	0.90	0.93	0.80	0.83	0.81	0.86	0.87	0.86	0.83
aeroBERT-NER$_{LC}$	0.81	**0.92**	0.86	0.88	0.88	0.88	0.97	**0.98**	0.97	0.90	**0.92**	0.91	0.83	0.87	0.85	0.90
aeroBERT-NER$_{BU}$	0.67	0.75	0.71	0.58	0.74	0.65	0.95	0.95	0.95	0.81	0.80	0.80	0.87	0.89	0.88	0.83
aeroBERT-NER$_{BC}$	**0.88**	0.88	**0.88**	**0.98**	**0.92**	**0.95**	**0.98**	**0.98**	**0.98**	**0.93**	0.91	**0.92**	**0.89**	**0.91**	**0.90**	**0.92**

The training time for aeroBERT-NER$_{BC}$ and aeroBERT-NER$_{LC}$ were 268.6 and 873.9 seconds, respectively. For the rest of this section, aeroBERT-NER$_{BC}$ will be referred to as aeroBERT-NER since it was deemed to be the best-performing model out of all BERT variants that were fine-tuned for the NER task.

6.2.2 Glossary creation

One goal of this research is to facilitate the creation of a glossary after using aeroBERT-NER for identifying NEs in aerospace requirements. To test this capability, a separate test set containing 20 requirements was created. The requirements in the test set are provided in Table B.1 in Appendix B. These requirements are chosen from 14 CFR §25.1301 through §25.1360. aeroBERT-NER is first used to identify different NEs. A Python script is then used to concatenate the NE subwords (or WordPiece tokens) identified by aeroBERT-NER. These concatenated NEs are then used to create a glossary in which they are grouped following the category they belong to (SYS, ORG, VAL, DATETIME, and RES). A manual check is then performed to identify the NEs in the test set and compare them with the NEs that were identified by aeroBERT-NER and added to the glossary.

Out of the 32 SYS names, 20 SYS names were identified verbatim. Of the remaining SYS NEs, subwords of 5 system names were identified. 62.5% of all the SYS NEs were identified verbatim, however, the percentage goes up to 78.13% if the subwords are included as well (Table 6.3). Table 6.4 shows some of the NEs that were identified by aeroBERT-NER and added to the glossary. The NEs in this table are newly identified and the model was not trained on these specific NEs (systems names, etc.).

The test set did not contain any ORG NEs, which is typical of aerospace requirements. aeroBERT-NER did not falsely identify any ORGs, which stresses the model's robustness. Despite the sparse presence of ORG entities in requirements, it was included as a category, in case aeroBERT-NER is used on aerospace texts apart from

Table 6.3: Percentage of NEs identified by aeroBERT-NER in the test set

Type of NE	Test Set (total count)	aeroBERT-NER (verbatim)	aeroBERT-NER (including subwords)
SYS	32	20 (62.5%)	25 (78.13%)
RES	11	11 (100%)	-
DATETIME	2	1 (50%)	2 (100%)
ORG	0	-	-
VAL	0	-	-

Table 6.4: Example of NEs identified and added to the glossary

SYS	RES	DATETIME	ORG	VAL
flight deck controls	paragraph (b)(1)	December 1, 2012	-	-
equipment	Section 25-997	-	-	-
electrical system	Section 25-997(d)	-	-	-
direction indicator	Section 25-1019	-	-	-
nonstabilized magnetic compass	Section 25-1019(a)(2)	-	-	-
instrument	paragraph (a)	-	-	-
airplanes	Section 21-16	-	-	-
powerplant instrument	Section 25-1303	-	-	-
magnetic direction indicator	Section 25-1309	-	-	-
autopilot	Sections 25-993	-	-	-
autothrust	25-1183	-	-	-
...	-	-	-	-

requirements. Similarly, no VAL NEs occurred in the requirements in the test set and as a result, none were identified by the model. Out of the two DATETIME NEs that occurred in the test set, one was identified verbatim, whereas only subwords of the other entity were identified. Lastly, 100% of the RES NEs occurring in the test set were identified verbatim.

The creation of a glossary helps automatically determine the systems that the text or requirement is referring to. For example, the glossary in Table 6.4 indicates the requirements in the test set are about flight deck controls, electrical system, autopilot system, etc. In addition, the resources being referred to in the requirements are also being identified, which helps in narrowing down the resources of interest in case someone wants to dig further to get more context.

6.2.3 Comparison between aeroBERT-NER and $\text{BERT}_{\text{BASE}}$-NER

aeroBERT-NER is capable of properly identifying five types of NEs commonly found in aerospace texts and requirements. $\text{BERT}_{\text{BASE}}$-NER, on the other hand, is capable of identifying four types of NEs, which are not specific to the aerospace domain. The only common category between both the LMs is ORG.

Table 6.5: Comparing the characteristics of aeroBERT-NER with $\text{BERT}_{\text{BASE}}$-NER on aerospace text

Characteristics	aeroBERT-NER	$\text{BERT}_{\text{BASE}}$-NER
Model Information	$\text{BERT}_{\text{BASE}}$ cased fine-tuned on aerospace text and requirements	$\text{BERT}_{\text{BASE}}$ cased fine-tuned on English version of standard CoNLL-2003 NER
Types of NEs	5 [SYS, RES, DATETIME, **ORG**, VAL]	4 [PER, **ORG**, LOC, MISC]
% of NE identified	71% (32 out of 45)	0%
Words/sub-words identified	flight deck controls, autopilot, etc. (Table 6.4)	Section 25, Section

To compare the performance of both models on aerospace text in the test dataset, the type of NEs was dropped and the absolute number of entities that were identified by each model was considered since both models are not intended to identify the same types of NEs. The entities identified by each of the models were then compared with the entities manually identified during the glossary creation step.

aeroBERT-NER was able to identify 71% (32 out of 45) of the relevant NEs (not including the subwords), a sample of which is shown in Table 6.4. However, $\text{BERT}_{\text{BASE}}$-NER was unable to identify any NEs apart from two subwords ("Section 25", "Section"). Both of the identified subwords are tagged as MISC. A summary of the comparison is shown in Table 6.5. This illustrates the superior performance of aeroBERT-NER when compared to $\text{BERT}_{\text{BASE}}$-NER on aerospace text despite being fine-tuned on a small annotated NE aerospace corpus. This showcases the potential of transfer learning in the realm of NLP.

6.3 aeroBERT-Classifier

aeroBERT-Classifier was developed to classify aerospace requirements into three classes, namely, design, functional, and performance (Table 5.12). This section discusses the performance of aeroBERT-Classifier on the test set. In addition, the performance of this model is compared to other classification models, including fine-tuned GPT-2, trained Bi-LSTM (with GloVe) [132], which combines the power of pre-trained word embeddings and Bi-LSTM to capture context, and zero-shot model bart-large-mnli [39].

6.3.1 aeroBERT-Classifier Performance

aeroBERT-Classifier was trained on a dataset containing 279 requirements. The dataset was imbalanced, meaning there was more of one type of requirement as compared to others (136 design requirements compared to 89 functional, and 54 performance requirements). Accuracy as a metric will favor the majority class, design requirements in our case. Therefore, validation precision, recall, and F1 score were used to measure the model performance more rigorously. Table 6.6 provides the aggregate values for these metrics along with the breakdown for each requirement type.

The aeroBERT-Classifier$_{\text{LU}}$ (obtained by fine-tuning BERT$_{\text{LARGE-UNCASED}}$) was able to identify 100% (Recall) of functional requirements present in the test set, however, of all the requirements that the model identified as functional requirements, only 83% (Precision) belonged to this category. Similarly, aeroBERT-Classifier$_{\text{BC}}$ (obtained by fine-tuning BERT$_{\text{BASE-CASED}}$) was able to identify 80% of all the functional requirements and 75% of all the performance requirements present in the test set. The precision obtained for functional and performance requirements were 0.89 and 0.86, respectively.

Table 6.6: Requirements classification results on aerospace requirements dataset. The highest score for each metric per class is shown in **bold**. (P = Precision, R = Recall, F1 = F1 Score)

Models	Design			Functional			Performance			Avg.
	P	R	F1	P	R	F1	P	R	F1	F1
aeroBERT-Classifier$_{\mathbf{LU}}$	**0.91**	0.77	0.83	0.83	**1.0**	**0.91**	0.75	0.75	0.75	**0.83**
aeroBERT-Classifier$_{\mathbf{LC}}$	0.86	**0.92**	**0.89**	0.82	0.90	0.86	0.83	0.63	0.71	0.82
aeroBERT-Classifier$_{\mathbf{BU}}$	0.80	**0.92**	0.86	**0.89**	0.80	0.84	**0.86**	0.75	**0.80**	**0.83**
aeroBERT-Classifier$_{\mathbf{BC}}$	0.79	0.85	0.81	0.80	0.80	0.80	**0.86**	0.75	**0.80**	0.80
GPT-2	0.67	0.60	0.63	0.67	0.67	0.67	0.70	**0.78**	0.74	0.68
Bi-LSTM (GloVe)	0.75	0.75	0.75	0.75	0.60	0.67	0.43	0.75	0.55	0.68
bart-large-mnli	0.43	0.25	0.32	0.38	0.53	0.44	0.0	0.0	0.0	0.34

Of all the variants of aeroBERT-Classifier evaluated, *uncased* variants outperformed the *cased* variants. This indicates that "casing" is not as important to text classification as it would be in the case of a NER task. In addition, the overall average F1 scores obtained by aeroBERT-Classifier$_{\text{LU}}$ and aeroBERT-Classifier$_{\text{BU}}$ were the same. This suggests that the *base-uncased* model is capable of learning the desired patterns in aerospace requirements in less training time and, hence is preferred.

Various iterations of the model training and testing were performed, and the model performance scores were consistent. In addition, the aggregate precision and recall were not very far off from each other, giving rise to a high F1 score (harmonic mean of precision and recall). Since the difference between the training and test performance is low despite the small size of the dataset, it is expected that the model will generalize well to unseen requirements belonging to the three categories.

Table 6.7 provides a list of requirements from the test set that were misclassified (Predicted label $\neq$ Actual label) by aeroBERT-Classifier$_{\text{BU}}$. A confusion matrix summarizing the classification task is shown in Figure 6.3. It is important to note that some of the requirements were difficult to classify even by SMEs with expertise in requirements engineering.

The test set contained 13 design, 10 functional, and 8 performance requirements (Table 5.12). As seen in Table 6.7 and Figure 6.3, out of the 13 design requirements, only one was misclassified as a performance requirement. Of the 8 performance requirements, 2 were misclassified. And 2 of the 10 functional requirements were misclassified.

The training and testing were carried out multiple times, and the requirements shown in Table 6.7 were consistently misclassified, which might have been due to ambiguity in the labeling. Hence, it is important to have a *human-in-the-loop* (preferably a Subject Matter Expert (SME)) who can make a judgment call on whether a certain requirement was labeled wrongly or to support a requirement rewrite to resolve

Table 6.7: List of requirements (from test set) that were misclassified by aeroBERT-Classifier$_{\text{BU}}$ (0: Design; 1: Functional; 2: Performance)

Requirements	Actual	Predicted
The installed powerplant must operate without any hazardous characteristics during normal and emergency operation within the range of operating limitations for the airplane and the engine.	2	1
Each flight recorder must be installed so that it remains powered for as long as possible without jeopardizing emergency operation of the airplane.	0	2
The microphone must be so located and, if necessary, the preamplifiers and filters of the recorder must be so adjusted or supplemented, so that the intelligibility of the recorded communications is as high as practicable when recorded under flight cockpit noise conditions and played back.	2	0
A means to extinguish fire within a fire zone, except a combustion heater fire zone, must be provided for any fire zone embedded within the fuselage, which must also include a redundant means to extinguish fire.	1	0
Thermal/acoustic materials in the fuselage, must not be a flame propagation hazard.	1	0

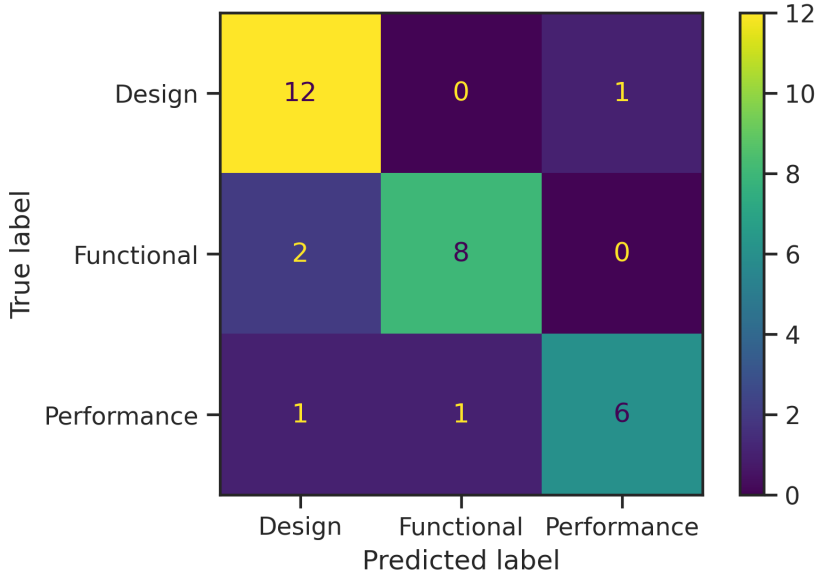


Figure 6.3: Confusion matrix showing the breakdown of the true and predicted labels by the aeroBERT-Classifier on the test data

ambiguities.

6.3.2 Comparison between aeroBERT-Classifier and other text classification LMs

aeroBERT-Classifier was compared to other LMs capable of text classification with the performance metrics for each of the LMs summarized in Table 6.6. aeroBERT-Classifier and GPT-2 were fine-tuned and tested on aerospace requirements. Bi-LSTM (GloVe) was trained and tested on aerospace requirements from scratch. Lastly, an off-the-shelf ZSL classifier (bart-large-mnli) was used for classifying aerospace requirements without being trained/fine-tuned beforehand.

In the field of aerospace requirements engineering, where labeled datasets are often limited, transfer learning can become an essential tool for developing generalizable models for downstream tasks such as requirements classification. The performance of BERT, GPT-2, and Bi-LSTM on the small requirements dataset used for fine-tuning/training in this study highlights the effectiveness of transfer-learning approaches in this domain (Table 6.6). Despite the limited availability of labeled data, these models were able to outperform bart-large-mnli on in-domain text, emphasizing the importance of transfer learning for achieving generalizable performance in NLP tasks with limited labeled data. The remainder of this section provides a one-on-one comparison between aeroBERT-Classifier$_{\text{BC}}$ and bart-large-mnli.

aeroBERT-Classifier is capable of classifying requirements into three types, as shown in Table 5.12. *bart-large-mnli*, on the other hand, is capable of classifying sentences into provided classes using Natural Language Inference (NLI)-based zero-shot text classification [85].

All the requirements present in the test set were classified using bart-large-mnli to facilitate the comparison with the aeroBERT-Classifier. The names of the types of requirements (design requirement, functional requirement, and performance requirement) were provided to the model for zero-shot text classification.

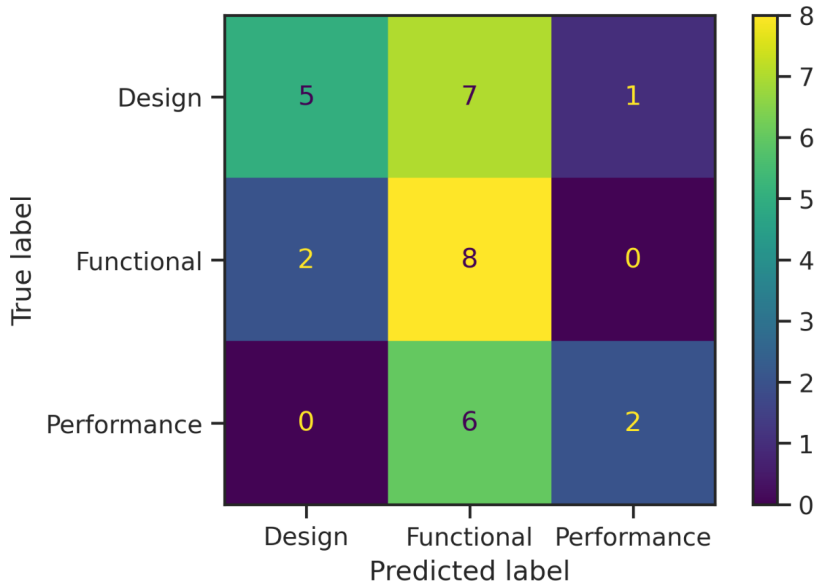


Figure 6.4: Confusion matrix showing the breakdown of the true and predicted labels by the bart-large-mnli model on the test data

Figure 6.4 shows the true and the predicted labels for all the requirements in the test set by bart-large-mnli. Upon comparing Figure 6.3 to Figure 6.4, aeroBERT-Classifier was able to correctly classify 83.87% of the requirements as compared to 43.39% when bart-large-mnli was used. The latter model seemed to be biased towards classifying most of the requirements as functional requirements. Had bart-large-mnli classified all the requirements as functional, the zero-shot classifier would have rightly classified 32.26% of the requirements. This illustrates the superior performance of the aeroBERT-Classifier despite it being trained on a small labeled dataset. Hence, while bart-large-mnli performs well on more general tasks like sentiment analysis, classification of news articles into genres, etc., using zero-shot classification, its performance is degraded in tasks involving specialized and structured texts such as aerospace requirements.

It is important to have an SME (Subject Matter Expert) overseeing the requirement classification process, who can make a decision on whether a requirement is being mislabeled or misclassified. Additionally, if necessary, requirements can be rewritten to eliminate any ambiguities.

6.4 Standardization of requirements

A two-pronged approach was employed for the standardization of requirements, namely, the creation of a requirements table, and the identification of boilerplates. This section discusses the results obtained upon applying the methodologies developed in the previous chapter, to NL requirements to standardize them.

6.4.1 Creation of requirements table

Table 6.8 shows a requirement table with five requirements belonging to various types and their corresponding properties. The various columns of the table were populated by extracting information from the original requirement text using different LMs (aeroBERT-NER and aeroBERT-Classifier) that were fine-tuned on an aerospace-specific corpus. This table can be exported as an Excel spreadsheet, which can then be verified by a subject matter expert (SME) and any missing information can be added.

The creation of a requirement table, as described above, is an important step toward the standardization of requirements by aiding the creation of tables and models in SysML. The use of various LMs automates the process and limits the manual way of populating the table. In addition, aeroBERT-NER, and aeroBERT-Classifier generalize well and are capable of identifying named entities and classifying requirements despite the noise and variations that can occur in NL requirements. This methodology for extracting information from NL requirements and storing them in tabular format triumphs in comparison to using a dictionary-based approach which needs constant updating as the requirements evolve.

Table 6.8: Requirement table containing columns extracted from NL requirements using language models. This table can be used to aid the creation of SysML requirement table.

Serial No.	Name	Text	Type of Requirement	Property	Related To
1	nozzle guide vanes	All nozzle guide vanes should be weld repairable without a requirement to strip coating.	Design	-	{SYS: [nozzle guide vanes]}
2	flight recorder	The state estimates supplied to the flight recorder shall meet the aircraft level system requirements and the functionality specified in Section 23-2500.	Design	{RES: [Section 23-2500]}	{SYS: [flight recorder, aircraft]}
3	pressurized airplanes	Pressurized airplanes with maximum operating altitude greater than 41000 feet, must be capable of detecting damage to the pressurized cabin structure before the damage could result in rapid decompression that would result in serious or fatal injuries.	Functional	{VAL: [greater than, 41000 feet]}	{SYS: [pressurized airplanes, pressurized cabin structure]}
4	fuel system	Each fuel system must be arranged so that any air which is introduced into the system will not result in power interruption for more than 20 seconds for reciprocating engines.	Performance	{VAL: [20 seconds]}	{SYS: [fuel system, reciprocating engines]}
5	structure	The structure must be able to support ultimate loads without failure for at least 3 seconds.	Performance	{VAL: [3 seconds]}	{SYS: [structure]}

6.4.2 Identification of boilerplates by examining patterns in requirements text

The requirements were first classified into various types using the aeroBERT-Classifier. Boilerplate templates for various types of requirements were then determined by utilizing sentence chunks and named entities to detect patterns. To account for the diversity of these requirements, multiple templates were recognized for each type.

Table 6.9: Summary of boilerplate template identification task. Two, five, and three boilerplate templates were identified for Design, Functional, and Performance requirements that were used for this study.

Requirement type	Count	Boilerplate Count	% of requirements covered
Design (0)	149	2	∼55%
Functional (1)	100	5	63%
Performance (2)	61	3	∼58%

Table 6.9 shows a breakdown of the number of boilerplate templates that were identified for each requirement type and the percentage of requirements covered by the boilerplate templates. Two boilerplates were identified for design requirements that were used for this study. Five and three boilerplates were identified for Functional and Performance requirements respectively. A greater variability was observed in the textual patterns occurring in Functional requirements which resulted in a greater number of boilerplates for this particular type. The identified templates are discussed in detail in the following subsections.

Design Requirements

In analyzing the design requirements as they were presented, it was discovered that two separate boilerplate structures were responsible for roughly ∼55% of the requirements used in the study. These two structures were able to encompass the majority of the requirements, and incorporating additional boilerplate templates would have resulted in overfitting them to only a handful of requirements each. This would have compromised their ability to be applied broadly, reducing their overall generalizabil-

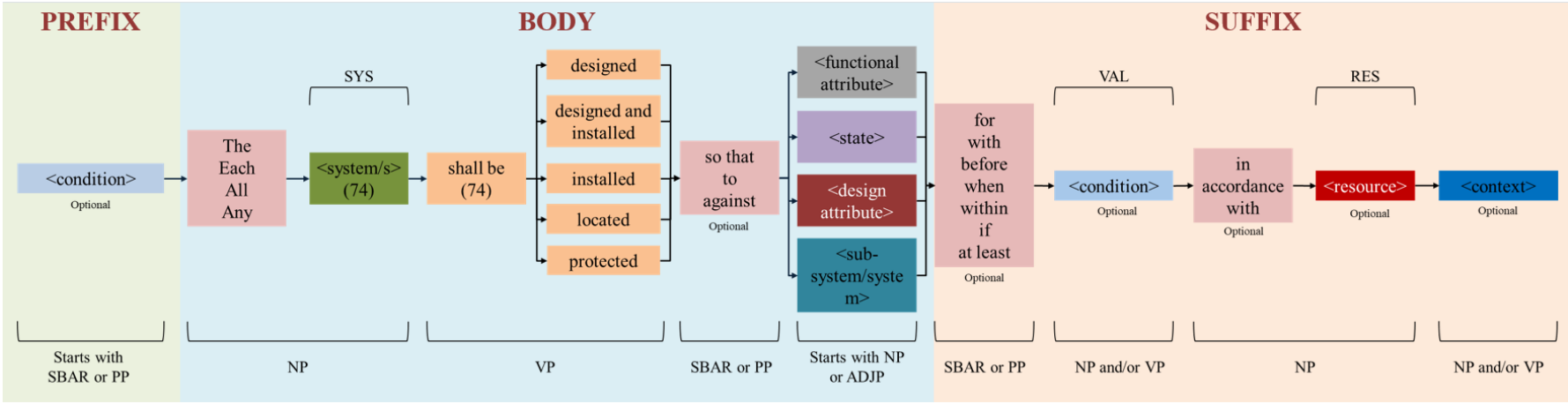
ity.

The first boilerplate is shown in Figure 6.5 and focuses on requirements that mandate the "way" a system should be designed and/or installed, its location, and whether it should protect another system/sub-system from a certain ⟨condition⟩ or ⟨state⟩. The NE and sentence chunk tags are displayed above and below the boilerplate structure. Based on these tags, it was observed that a requirement with a ⟨condition⟩ in the beginning usually starts with a prepositional phrase (PP) or subordinate clause (SBAR). This is followed by a noun phrase (NP), which contains a ⟨system⟩ name, which can be distinctly identified within the NP by using the NE tag (SYS). The initial NP is always succeeded by a verb phrase (VP) which contains the term "shall [variations]". These "[variations]" (shall be designed, shall be protected, shall have, shall be capable of, shall maintain, etc.) were crucial for the identification of different types of boilerplates since they provided information regarding the action that the ⟨system⟩ should be performing (to protect, to maintain, etc.).

In the case of Figure 6.5, the observed "[variations]" were *be designed, be designed and installed, installed, located, and protected* respectively. The VP is followed by a subordinate clause (SBAR) or prepositional phrase (PP) but this is optional. This is then followed by a NP or adjective clause (ADJP) and can contain either a ⟨functional attribute⟩, ⟨state⟩, ⟨design attribute⟩, or a ⟨sub-system/system⟩. This brings an end to the main *Body* of the requirement. The *Suffix* is optional and can contain additional information such as operating and environmental conditions, resources, and context.

The second boilerplate for design requirements is shown in Figure 6.6. This boilerplate accounts for the design requirements that mandate a certain ⟨functional attribute⟩ that a system should have, a ⟨sub-system/system⟩ it should include, and any ⟨design attribute⟩ it should have by design. Similar to the previous boilerplate, the NEs and sentence chunk tags are displayed above and below the structure.

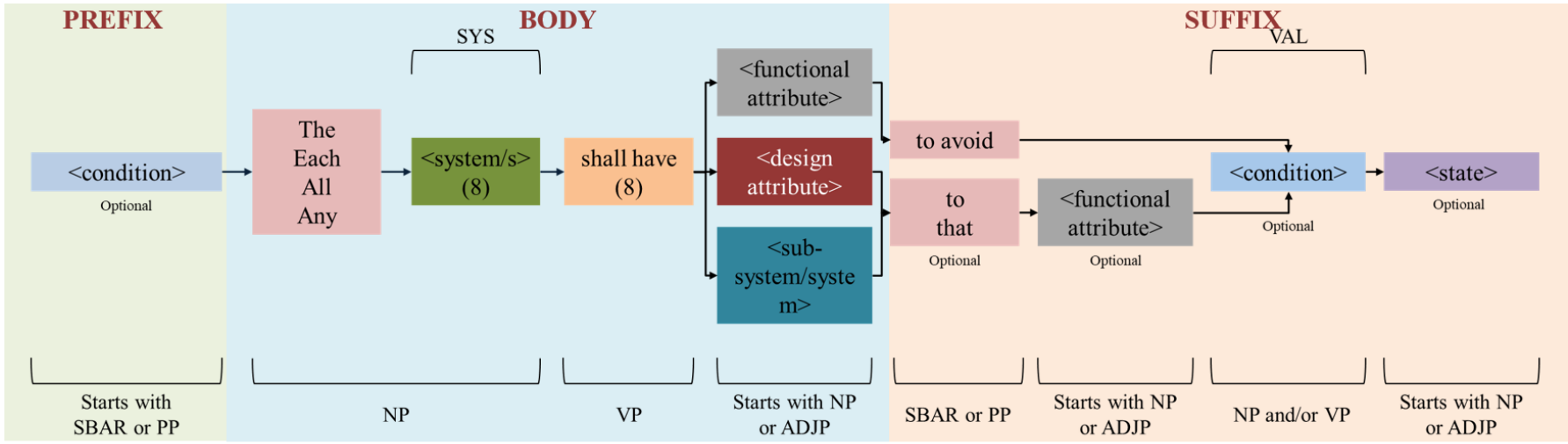
The rest of the design requirements were examined, however, no common patterns



The control system shall be designed for pilot forces applied in the same direction, using individual pilot forces not less than 0.75 times those obtained in accordance with Section 25-395.

Any taxi and landing lights shall be designed and installed so they provide sufficient light for night operations.

Figure 6.5: The schematics of the first boilerplate for **design requirements** along with some examples that fit the boilerplate are shown here. This boilerplate accounts for 74 of the 149 design requirements (∼50%) used for this study and is tailored toward requirements that mandate the way a ⟨system⟩ should be designed and/or installed, its location, and whether it should protect another ⟨system/sub-system⟩ given a certain ⟨condition⟩ or ⟨state⟩. Parts of the NL requirements shown here are matched with their corresponding boilerplate elements via the use of the same color scheme. In addition, the sentence chunks and named entity (NEs) tags are displayed below and above the boilerplate structure respectively.



Each part of the airplane shall have adequate provisions for ventilation and drainage.

Each recorder container shall have reflective tape affixed to its external surface to facilitate its location under water.

Each recorder container shall have an underwater locating device.

Figure 6.6: The schematics of the second boilerplate for **design requirements** along with some examples that fit the boilerplate are shown here. This boilerplate accounts for 8 of the 149 design requirements (∼5%) used for this study and focuses on requirements that mandate a ⟨functional attribute⟩ , ⟨design attribute⟩ , or the inclusion of a ⟨system/subsystem⟩ by design. Two of the example requirements highlight the ⟨design attribute⟩ element which emphasizes additional details regarding the system design to facilitate a certain function. The last example shows a requirement where a ⟨subsystem⟩ is to be included in a system by design.

were observed in most of them to warrant boilerplate creation specific to these requirements. Boilerplates, if created, would have fewer requirements compatible with them, which could have undermined their capacity to be applied more generally. As a result, the overall generalizability of the templates would have been reduced.

Functional Requirements

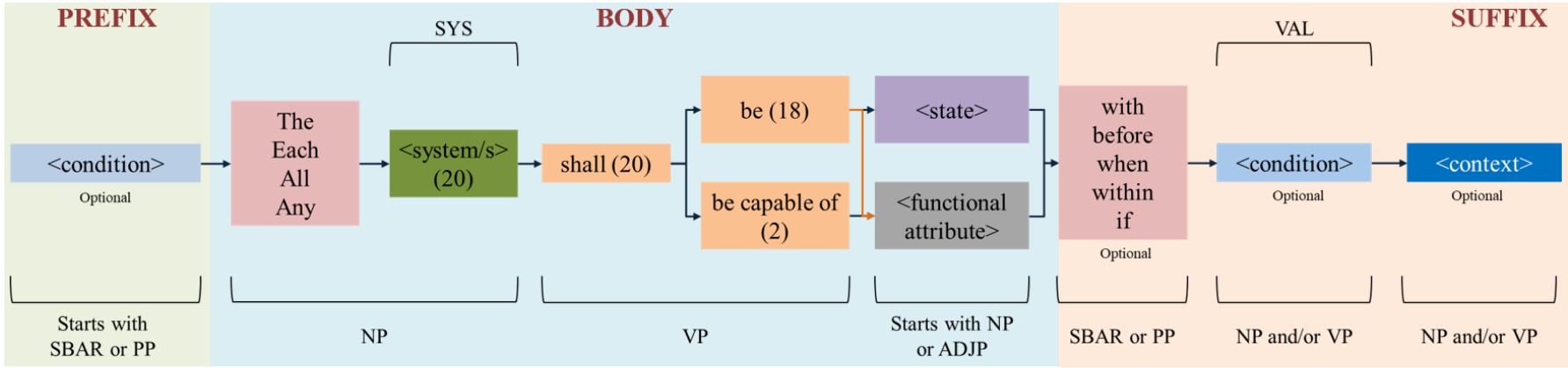
In analyzing the NL functional requirements as they appeared in Parts 23 and 25 of Title 14 CFRs, the study identified five separate boilerplate structures that encompassed a total of 63% of the functional requirements. However, introducing more boilerplate templates would have led to fitting a smaller number of requirements to these structures, potentially limiting their overall applicability and generalizability.

The first boilerplate is shown in Figure 6.7 and is tailored toward requirements that describe the capability of a ⟨system⟩ to be in a certain ⟨state⟩ or perform a certain ⟨function⟩.The example requirement (especially 1) focuses on the handling characteristics of the system (airplane in this case). The associated sentence chunks and NEs for each of the elements of the boilerplate are also shown.

The second boilerplate for functional requirements is shown in Figure 6.8 and focuses on requirements that require the ⟨system⟩ to have a certain ⟨functional attribute⟩ or maintain a particular ⟨state⟩. This boilerplate structure accounts for 15% of all the functional requirements.

Figure 6.9 shows the third boilerplate for functional requirements and is tailored toward requirements that require the ⟨system⟩ to protect another ⟨sub-system/system⟩ or ⟨user⟩ against a certain ⟨state⟩ or another ⟨sub-system/system⟩. This boilerplate structure accounts for 7% of all the functional requirements.

Figure 6.10 shows the fourth boilerplate for functional requirements and is tailored toward requirements that require the ⟨system⟩ to provide a certain ⟨functional

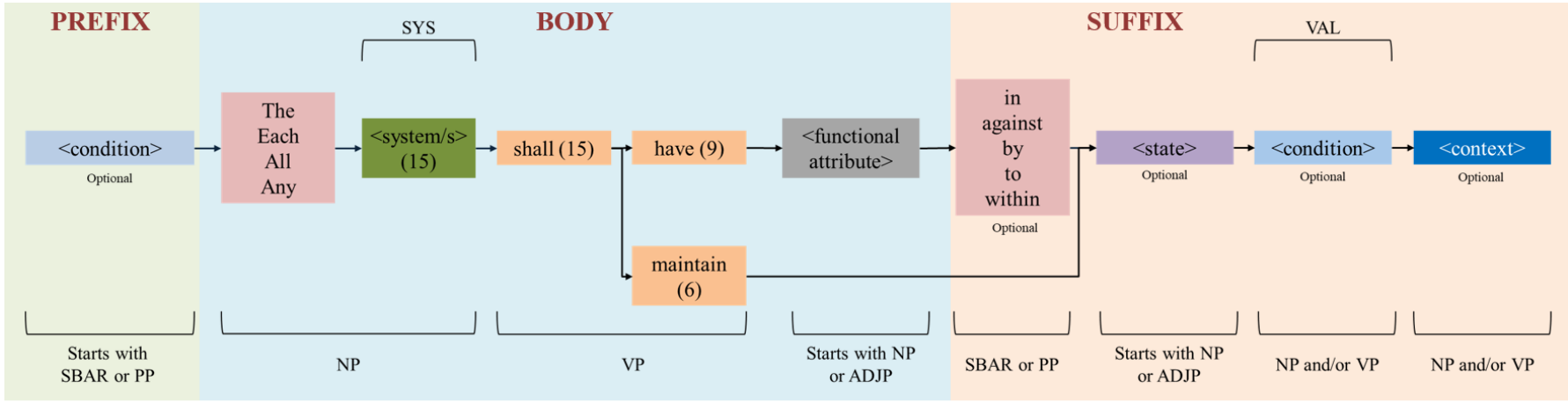


The airplane shall be controllable and maneuverable, without requiring exceptional piloting skill, alertness, or strength, within the operating envelope at all loading conditions for which certification is requested.

The insulation on electrical wire and electrical cable shall be self-extinguishing.

All pressurized airplanes shall be capable of continued safe flight and landing following a sudden release of cabin pressure.

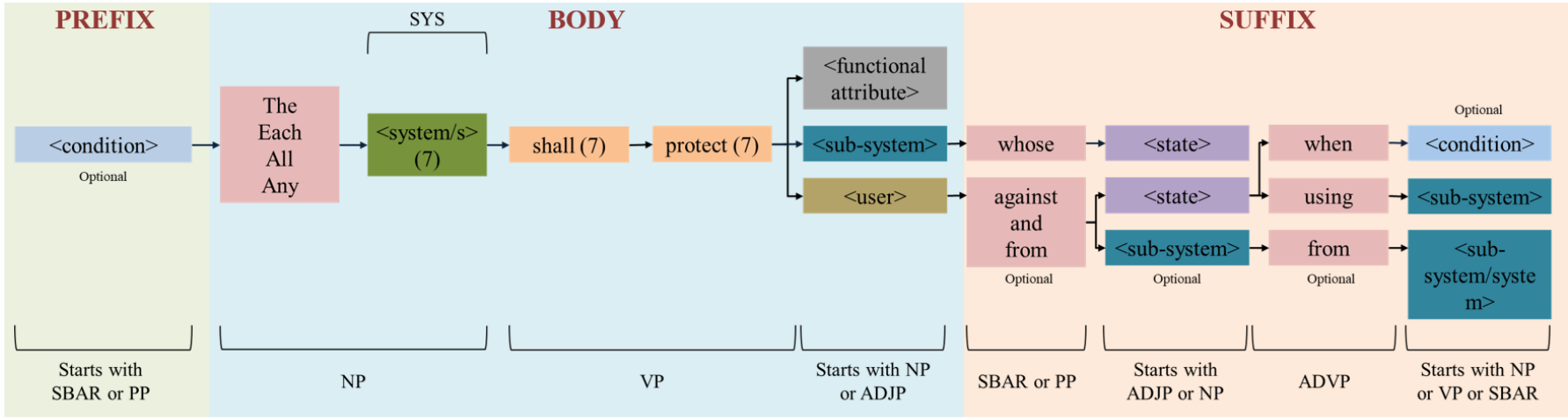
Figure 6.7: The schematics of the first boilerplate for **functional requirements** along with some examples that fit the boilerplate is shown here. This boilerplate accounts for 20 of the 100 functional requirements (20%) used for this study and is tailored toward requirements that describe the capability of a ⟨system⟩ to be in a certain ⟨state⟩ or have a certain ⟨functional attribute⟩. The example requirement (especially 1) focuses on the handling characteristics of the system (airplane in this case).



The airplane shall maintain longitudinal trim without further force upon, or movement of, the primary flight controls.

With the cabin configured for takeoff or landing, the airplane shall have means of egress, that can be readily located and opened from the inside and outside.

Figure 6.8: The schematics of the second boilerplate for **functional requirements** along with some examples that fit the boilerplate is shown here. This boilerplate accounts for 15 of the 100 functional requirements (15%) used for this study and is tailored toward requirements that require the ⟨system⟩ to have a certain ⟨functional attribute⟩ or maintain a particular ⟨state⟩.



The airplane design shall protect the pilot and flight controls from propellers.

Each baggage and cargo compartment shall protect any controls, wiring, lines, equipment, or accessories whose damage or failure would affect safe operations.

The trim systems shall protect against inadvertent, incorrect, or abrupt trim operation.

Figure 6.9: The schematics of the third boilerplate for **functional requirements** along with some examples that fit the boilerplate are shown here. This boilerplate accounts for 7 of the 100 functional requirements (7%) used for this study and is tailored toward requirements that require the ⟨system⟩ to protect another ⟨sub-system/system⟩ or ⟨user⟩ against a certain ⟨state⟩ or another ⟨sub-system/system⟩.

attribute⟩ given a certain ⟨condition⟩. This boilerplate structure accounts for 15% of all the functional requirements.

Figure 6.11 shows the fifth boilerplate for functional requirements and is specifically focused on requirements related to the *cockpit voice recorder* since a total of six requirements in the entire dataset were about this particular system and its ⟨functional attribute⟩ given a certain ⟨condition⟩. This boilerplate structure accounts for 6% of all the functional requirements. Although it is generally not recommended to have a boilerplate template that is specific to a particular system, in this case, it was deemed acceptable because a significant portion of the requirements pertained to that system, and the dataset used was relatively small.

Performance Requirements

Three distinct boilerplates were identified for performance requirements which accounted for a total of (∼58%) of all the requirements belonging to this type.

The first boilerplate for performance requirements is shown in Figure 6.12. This particular boilerplate has the element ⟨system attribute⟩, which is unique as compared to the other boilerplate structures. In addition, this boilerplate caters to the performance requirements specifying a ⟨system⟩ or ⟨system attribute⟩ to satisfy a certain function or ⟨condition⟩.∼33% of all the performance requirements match this template.

Figure 6.13 shows the second boilerplate for performance requirements. This boilerplate accounts for 12 of the 61 performance requirements (∼20%) used for this study. This boilerplate focuses on performance requirements that specify a ⟨functional attribute⟩ that a ⟨system⟩ should have or maintain given a certain ⟨state⟩ or ⟨condition⟩.

Lastly, Figure 6.14 shows the third boilerplate for performance requirements. This

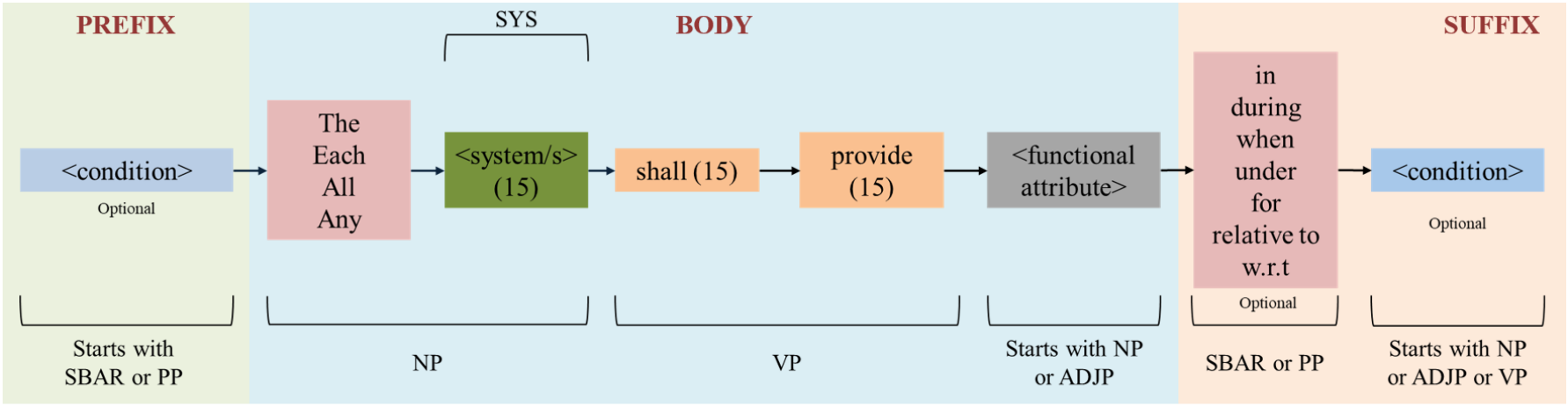
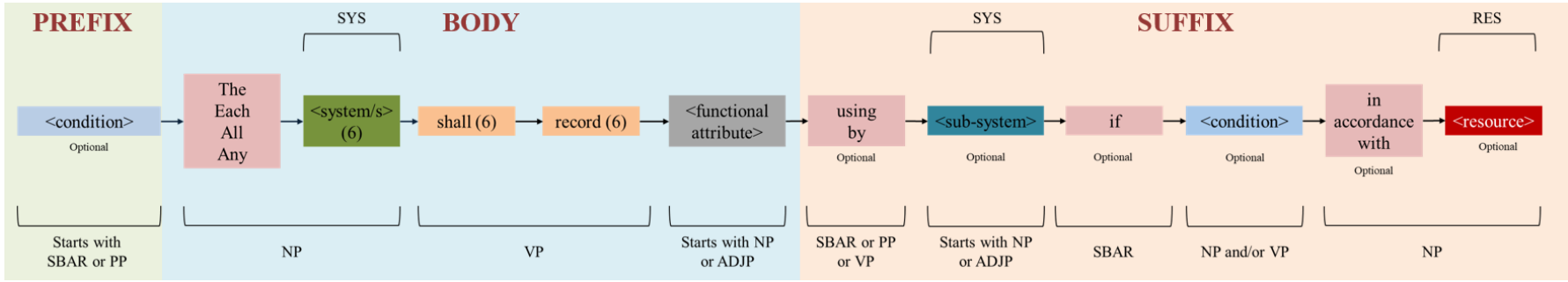


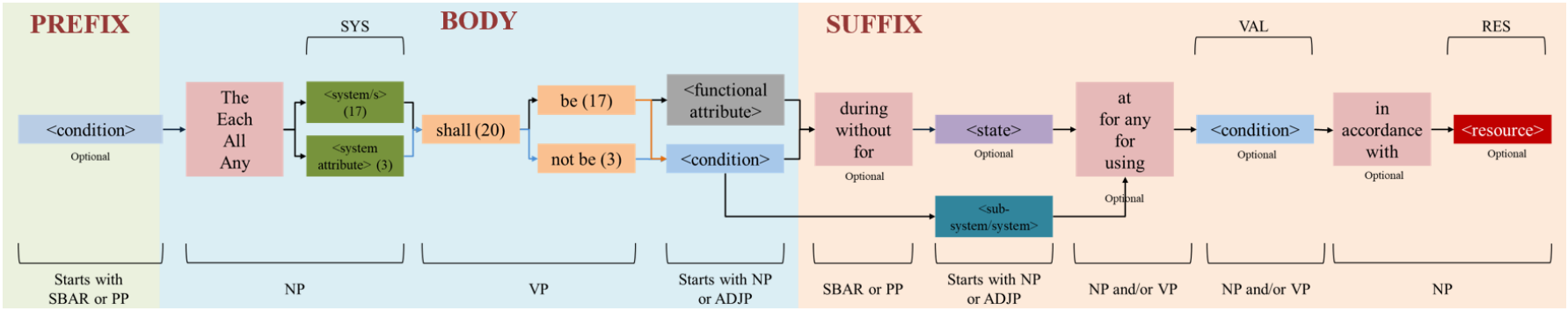
Figure 6.10: The schematics of the fourth boilerplate for **functional requirements** along with some examples that fit the boilerplate is shown here. This boilerplate accounts for 15 of the 100 functional requirements (15%) used for this study and is tailored toward requirements that require the ⟨system⟩ to provide a certain ⟨functional attribute⟩ given a certain ⟨condition⟩.



Each cockpit voice recorder shall record voice communications of flightcrew members on the flight deck, using the airplane's interphone system.

Each cockpit voice recorder shall record voice communications of flightcrew members using the passenger loudspeaker system, if there is such a system and if the fourth channel is available in accordance with the paragraph (c)(4)(ii) of this section.

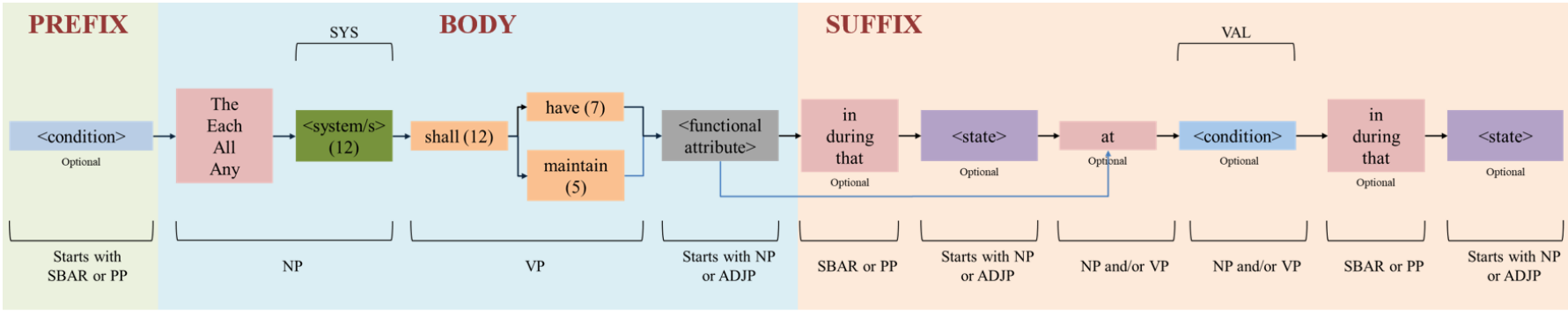
Figure 6.11: The schematics of the fifth boilerplate for **functional requirements** along with some examples that fit the boilerplate is shown here. This boilerplate accounts for 6 of the 100 design requirements (6%) used for this study and is specifically focused on requirements related to the *cockpit voice recorder* since a total of six requirements in the entire dataset were about this particular system and its ⟨functional attribute⟩ given a certain ⟨condition⟩.



The airplane shall be free from flutter, control reversal, and divergence at all speeds within and sufficiently beyond the structural design envelope.

The airplane's available gradient of climb shall not be less than 1.2 percent for two-engine airplane at each point along the takeoff path, starting at the point at which the airplane reaches 400 feet above the takeoff surface.

Figure 6.12: The schematics of the first boilerplate for **performance requirements** along with some examples that fit the boilerplate are shown here. This boilerplate accounts for 20 of the 61 performance requirements (∼33%) used for this study. This particular boilerplate has the element ⟨system attribute⟩ which is unique as compared to the other boilerplate structures. In addition, this boilerplate caters to the performance requirements specifying a ⟨system⟩ or ⟨system attribute⟩ to satisfy a certain ⟨condition⟩ or have a certain ⟨functional attribute⟩.



The airplane shall maintain longitudinal, directional, and lateral trim at 1.3 V_{SR1} during climbing flight with the remaining engines at maximum continuous power.

The airplane shall have controllable stall characteristics in straight flight, turning flight, and accelerated turning flight.

Figure 6.13: The schematics of the second boilerplate for **performance requirements** along with some examples that fit the boilerplate are shown here. This boilerplate accounts for 12 of the 61 performance requirements (∼20%) used for this study. This boilerplate focuses on performance requirements that specify a ⟨functional attribute⟩ that a ⟨system⟩ should have or maintain given a certain ⟨state⟩ or ⟨condition⟩.

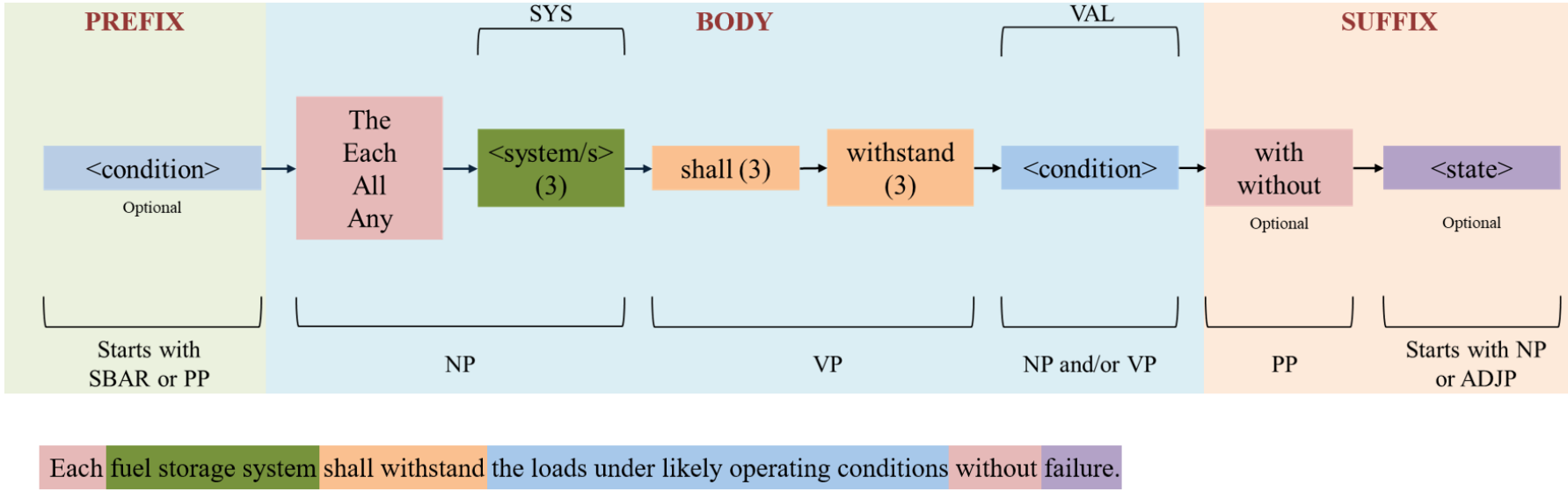


Figure 6.14: The schematics of the third boilerplate for **performance requirements** along with some examples that fit the boilerplate are shown here. This boilerplate accounts for 3 of the 61 performance requirements (∼5%) used for this study and focuses on a ⟨system⟩ being able to *withstand* and certain ⟨condition⟩ with or without ending up in a certain ⟨state⟩.

boilerplate accounts for 3 of the 61 performance requirements (∼5%) used for this study and focuses on a ⟨system⟩ being able to *withstand* a certain ⟨condition⟩ with or without ending up in a certain ⟨state⟩.

To summarize, the study found two boilerplate structures for design requirements, five for functional requirements, and three for performance requirements. A larger number of boilerplate structures were identified for functional requirements due to their greater variability. These structures were identified based on patterns observed in sentence chunks and named entities (NEs). The boilerplates can be utilized to create new requirements that follow the established structure or to assess the conformity of natural language requirements with the identified boilerplates. These activities are valuable for standardizing requirements on a larger scale and at a faster pace, and are expected to contribute to the adoption of Model-Based Systems Engineering (MBSE) in a more streamlined manner. Subject matter experts (SMEs) should review the identified boilerplates to ensure their accuracy and consistency.

CHAPTER 7
PRACTITIONER'S GUIDE

This chapter offers a condensed version of the methodologies (using flowcharts) devised in this dissertation, to facilitate the implementation by industry practitioners.

The chapter is structured into three main sections. The first section covers the development of language models. The second section discusses how the outputs from these models can be used to generate a requirements table. Finally, the third section demonstrates the creation of boilerplates through the use of three language models: aeroBERT-NER [120], aeroBERT-Classifier [126], and flair/chunk-english [114].

7.1 Creation of aeroBERT-NER and aeroBERT-Classifier

Figure 7.1 illustrates the process for developing aeroBERT-NER [120] and aeroBERT-Classifier [126], and provides references to the corresponding section numbers for more detailed information. To begin with, text pertaining to the aerospace domain was gathered, which was then used to construct two separate corpora: one containing definitions and other aerospace-related texts (the NER corpus), and the other containing only requirements (the requirements corpus). Both corpora were manually examined to identify the relevant named entities and requirement types for each.

To annotate the named entity corpus, individual ".txt" files were generated for each type of entity, allowing for easy differentiation. These files were utilized by a Python script that could match and tag the text accordingly. The annotation followed a BIO-tagging scheme to identify named entities. Likewise, the requirements within the requirements corpus were categorized and labeled according to their respective types. For instance, design requirements were labeled as '0', functional requirements were labeled as '1', and performance requirements were labeled as '2'. This completes

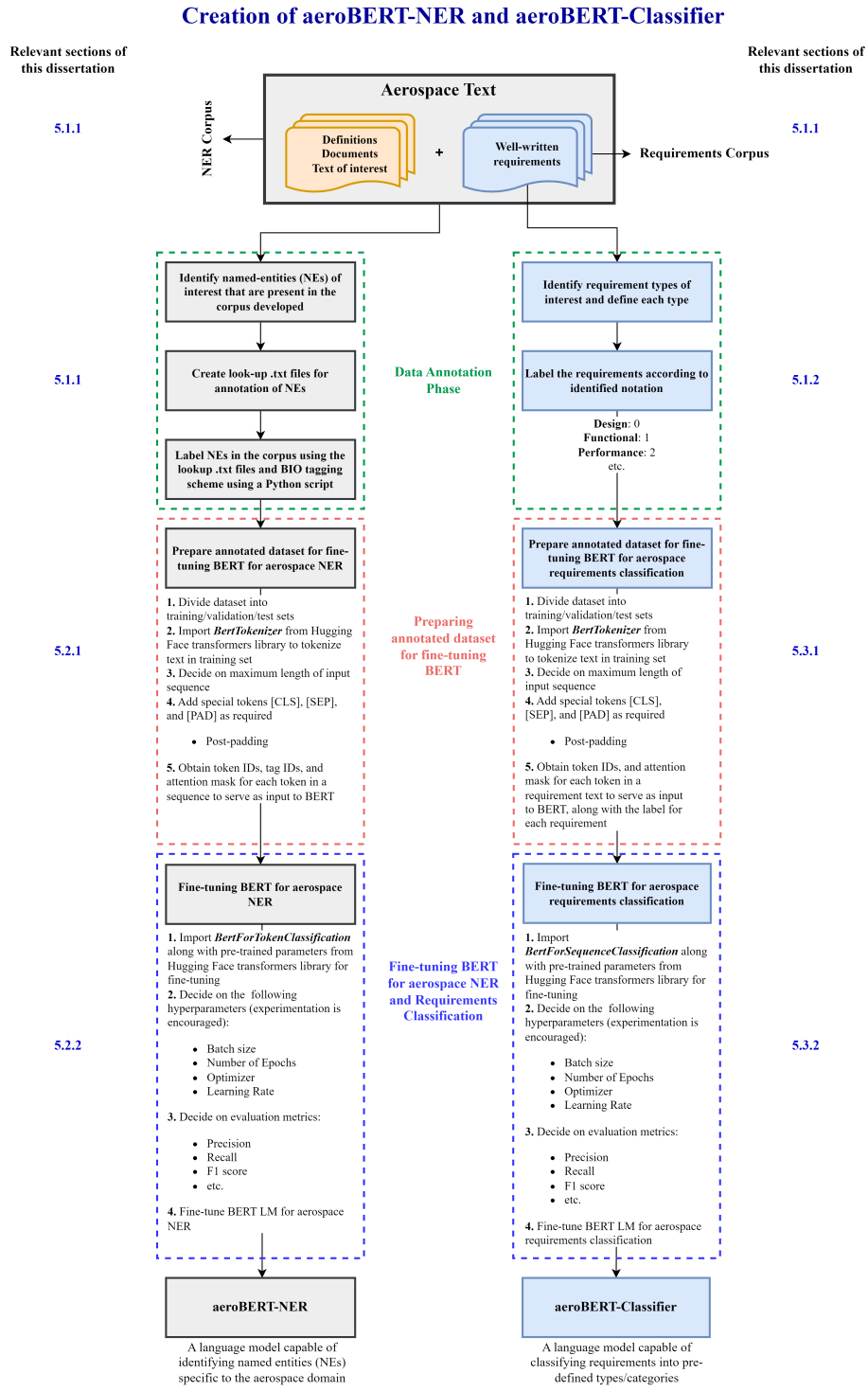


Figure 7.1: Practitioner's Guide to creation of aeroBERT-NER and aeroBERT-Classifier. A zoomed-in version of this figure can be found here.

the data annotation phase.

The NER corpus, which had been previously annotated, underwent pre-processing to prepare it for the fine-tuning of BERT for the identification of named entities within the aerospace domain. The corpus was split into training and validation sets, and the training set was tokenized with the BertTokenizer. To determine the optimal input sequence length, the distribution of sequence lengths within the training set was analyzed. Special tokens were added, and the sequences were post-padded. The resulting token IDs, tag IDs, and attention masks for each token were then generated as input to the BERT model. The same pre-processing steps were applied to the classification corpus, with the exception that only token IDs, attention masks, and labels for each requirement were utilized as inputs to the BERT model.

The `Transformers` library was used to import `BertForTokenClassification`. After selecting various hyperparameters, including the batch size, number of epochs, optimizer, and learning rate, the pre-trained parameters of BERT were fine-tuned using the annotated NER corpus (`archanatikayatray/aeroBERT-NER`) to create aeroBERT-NER. In a similar manner, `BertForSequenceClassification` was used to fine-tune BERT on the annotated aerospace requirements corpus (`archanatikayatray/aeroBERT-classification`), to obtain aeroBERT-Classifier. The models were assessed using evaluation metrics such as Precision, Recall, and F1 score.

aeroBERT-NER was trained in this study to recognize five named entity types (SYS, VAL, ORG, DATETIME, and RES), while aeroBERT-Classifier was trained to categorize requirements into three types (design, functional, and performance). Nonetheless, with adequate labeled training data, both models can be trained to identify additional types of named entities and requirements beyond those mentioned.

Findings:

1. The NER task yielded better results with the *cased* variants of aeroBERT-NER

compared to the *uncased* variants. This indicates that maintaining the case of the text is crucial for NER.

2. *Uncased* aeroBERT-NER variants outperformed the *cased* variants on the requirements classification task.

7.2 Creation of Requirements Table using language models

Figure 7.2 outlines the steps taken to create the requirements table, with references to relevant sections of the dissertation provided alongside. The process involves inputting requirements into both aeroBERT-Classifier and aeroBERT-NER, as shown in the flowchart for a single example, although multiple requirements can be processed simultaneously using both models.

For this thesis, a requirements table was created that has five columns: Name, Requirement Text, Type of Requirement, Property, and Related To. Further details on these columns can be found in Table 5.16. The initial entry in the *Name* column corresponds to the first system name (SYS named entity) detected by aeroBERT-NER [120]. The *Type of Requirement* column is filled in once aeroBERT-Classifier [126] classifies the requirement. The *Property* column is populated by a Python dictionary format that includes all the named entities (excluding SYS) identified by aeroBERT-NER, with the named entity type (RES, VAL, DATETIME, ORG) set as the dictionary key and the identified named entities presented as values in a Python list format. Lastly, the system name identified by aeroBERT-NER [120] is used to populate the *Related to* column.

The requirements table can be expanded with additional columns as necessary, although doing so may entail developing further language models capable of extracting the desired data. Moreover, aeroBERT-NER and aeroBERT-Classifier can be enhanced with additional named entities or requirement types to extract other pertinent information.

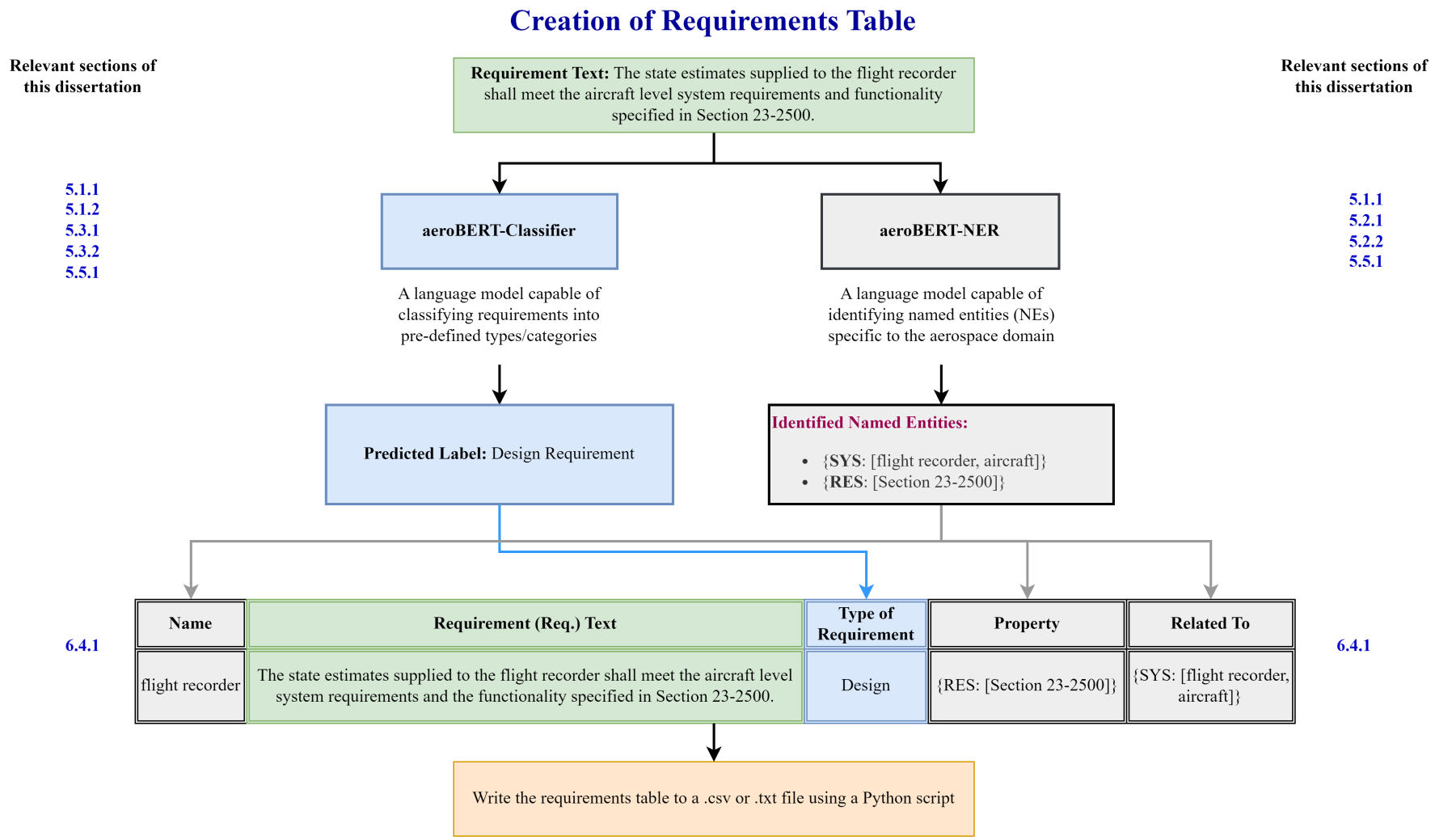


Figure 7.2: Practitioner's Guide to the creation of requirements table. A zoomed-in version of this figure can be found here.

7.3 Identification of Requirements boilerplates using language models

The flowchart in Figure 7.3 demonstrates the process of identifying boilerplate templates from well-written requirements. While the example shown in the flowchart illustrates the steps involved in using a single requirement, it is crucial to note that in order to generate boilerplate structures that can be applied more broadly, patterns observed across multiple requirements need to be identified. Moreover, for those who seek more information, the relevant section numbers from this dissertation are provided along with the flowchart.

The process of identifying boilerplate templates begins with the classification of requirements into different types utilizing aeroBERT-Classifier [126]. Next, text chunks, including Noun Phrases, Verb Phrases, etc., are identified using flair/chunk-english, and aerospace named entities are detected using aeroBERT-NER [120]. It is important to note that aeroBERT-NER is only capable of recognizing five types of named entities for which it was fine-tuned. Additionally, identifying elements in the NL requirements, such as ⟨condition⟩, ⟨system⟩, etc., is done manually because these elements may differ from one organization to another, or even within an organization. Once the elements have been identified, they are matched with the previously identified text chunks and named entities to partially automate the boilerplate identification process. This helps with the identification of boilerplate templates and with checking the conformance of requirements to the identified templates. The aggregation of various patterns in text chunks, named entities, and element sequences or their presence or absence is utilized to identify boilerplate templates.

Findings:

1. The determination of the threshold for the number of requirements that need to follow a specific pattern for it be considered a boilerplate template is at the

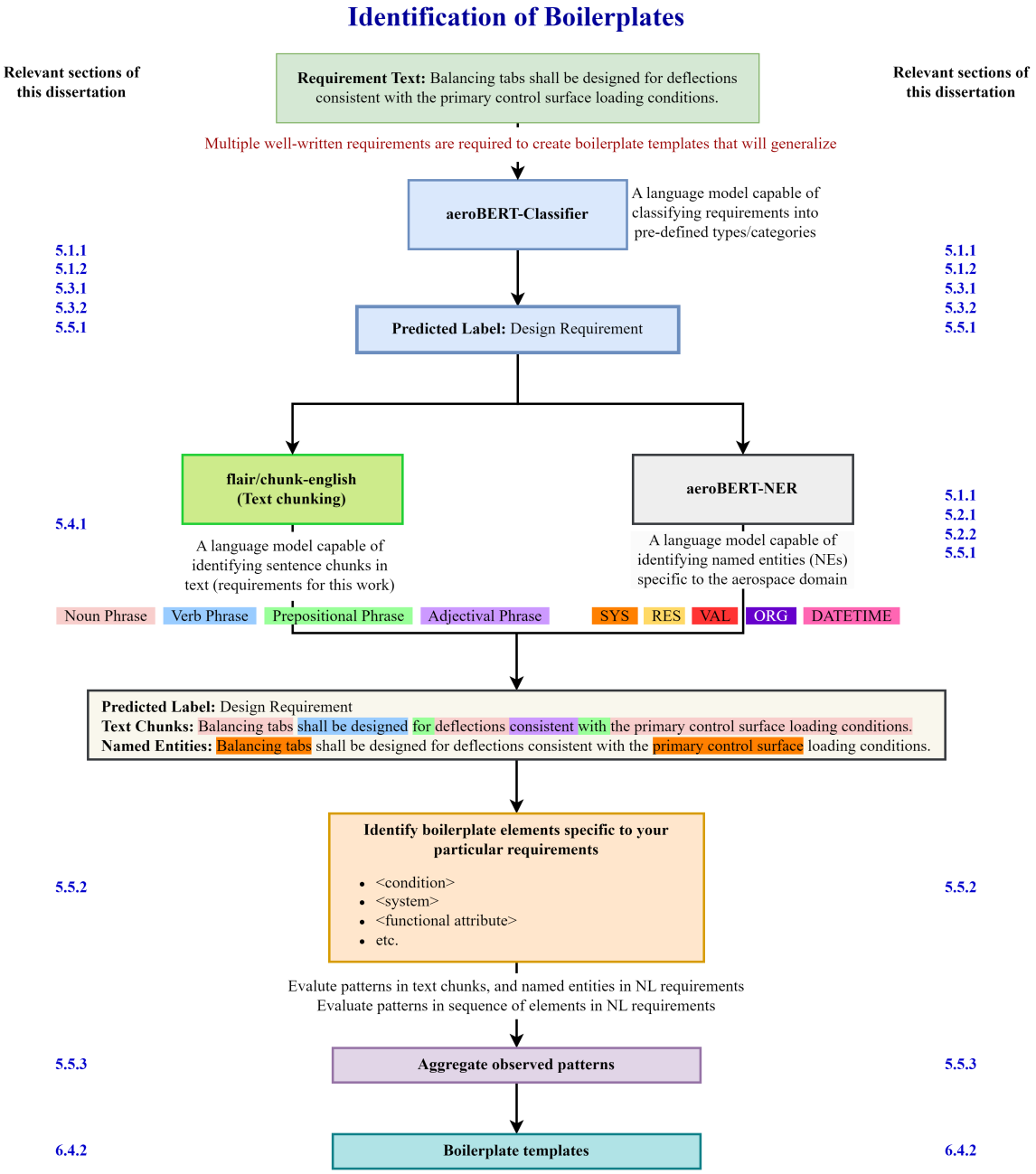


Figure 7.3: Practitioner's Guide to creation of aeroBERT-NER and aeroBERT-Classifier. A zoomed-in version of the figure can be found here.

discretion of the user. Typically, a useful guideline is to establish templates that correspond to a significant number of requirements in the requirements dataset.

2. Boilerplate structures may differ from one requirement type to another, and furthermore, there may be multiple boilerplate templates for each requirement

type.

The methodologies described above can standardize requirements and incorporate them into the requirements corpus for training the models, leading to improved robustness and larger corpus sizes over time.

CHAPTER 8
CONCLUDING REMARKS

8.1 Conclusions

The field of Natural Language Processing (NLP) has seen limited application in the aerospace industry, and its potential use in aerospace requirements engineering remains largely unexplored. Despite the crucial role of NL in various requirements engineering tasks throughout the system lifecycle, the aerospace industry has yet to fully exploit the potential of NLP in this area. This research aims to fill the gap in the literature by exploring the application of NLP techniques, including the use of LLMs, to aerospace requirements engineering.

Large Language Models (LLMs) have made it easier to use them in different domains than their original training through transfer learning. They are trained on extensive text corpora, and as a result, they possess a comprehensive understanding of language rules, which allows them to be fine-tuned on smaller labeled datasets for various downstream tasks, including specialized domains like medicine, law, engineering, etc. This is especially beneficial in domains with limited resources, such as aerospace requirements engineering, where acquiring large labeled datasets can be difficult due to the proprietary nature of the requirements and the subject matter expertise needed to create and annotate them. Therefore, the primary objective of this thesis was to develop and apply tools, techniques, and methodologies centered around LLMs that simplify the conversion of Natural Language (NL) requirements into semi-machine-readable requirements (Figure 1.7). The adoption of this approach is anticipated to promote the widespread utilization of LLMs for handling requirements on a larger scale and at a faster pace.

The certification requirements in Parts 23 and 25 of Title 14 CFR were the source of the requirements used in this study, as system requirements are generally proprietary. To familiarize readers with the requirements used in this study, numerous examples were included throughout this dissertation. Additionally, the annotated corpora employed for both the NER and classification tasks have been made open-source (refer to Appendix A), with the aim of supporting future research in this field.

The developed corpora were utilized to fine-tune the BERT LM for two distinct downstream tasks - identifying named entities specific to the aerospace domain, and classifying aerospace requirements into different types. Their performance was compared to that of other models, including some fine-tuned on the same corpora and off-the-shelf models. For NER and requirements classification on aerospace text, both models performed better than off-the-shelf models, despite being trained on small labeled datasets.

aeroBERT-NER and aeroBERT-Classifier have the ability to extract information from aerospace text and requirements and store them in a more accessible format, such as a Python list or dictionary. This extracted information was then utilized to showcase the methodology for creating a requirements table, which is a tabular format for storing requirements and their associated properties. This table can assist in the creation of a SysML requirements table, potentially reducing the time, resources, and manual labor involved in creating such a table from scratch.

Boilerplate templates for various types of requirements were identified using fine-tuned models for classification and named entity recognition, as well as an off-the-shelf sentence chunking model (`flair/chunk-english`). To account for variations within each type of requirement, multiple boilerplate templates were obtained. The use of these templates, particularly by inexperienced engineers working with requirements, will ensure that requirements are written in a standardized form from the beginning. In doing so, this dissertation democratizes a methodology for the identification of

boilerplates given a set of requirements, which can vary from one company or industry to another.

This dissertation presents a comprehensive methodology that outlines the collection of text, annotation, training, and validation of language models for aerospace text. Although the models developed may not be directly transferable to the proprietary system requirements used by aerospace companies, the methodology presented herein is still relevant and can be easily reproduced.

Finally, the benefits offered by using NLP for standardizing aerospace requirements contribute to speeding up the design and development process and reducing the workload on engineers. In addition, standardized requirements if/when made semi-machine readable support a model-centric approach to engineering.

8.2 Summary of Contributions

The dissertation resulted in the establishment of a methodology for producing annotated aerospace corpora and employing them to fine-tune language models. The key contributions are detailed below:

1. **Creation of first of its kind open-source annotated aerospace corpora:** Two different corpora (or datasets) were created as a part of this dissertation and can be accessed via the `Hugging Face` platform:

   (a) **`archanatikayatray/aeroBERT-NER`**: annotated aerospace named entity corpus

   (b) **`archanatikayatray/aeroBERT-classification`**: annotated aerospace requirements corpus

   The methodology used for data collection, cleaning, and annotation is described in detail to facilitate the reproducibility of corpus creation. The corpora generated in this study can be utilized for fine-tuning other language models for

downstream tasks, such as named entity recognition (NER) and requirements classification in the aerospace industry.

2. **Creation of language models for NER and requirements classification in the aerospace domain**:

   (a) **aeroBERT-NER**: Demonstration of using LMs to extract entities of interest from freeform NL aerospace requirements. This facilitates the conversion of NL requirements into information-rich data objects, which can be used for a variety of downstream tasks.

   (b) **aeroBERT-Classifier**: Demonstration of the viability of LMs in classifying high-level policy requirements such as the Federal Airworthiness Requirements (FARs), which have resisted earlier attempts to use NLP for automated processing. These requirements are particularly freeform and often complex expressions with a higher degree of classification difficulty than typical software or systems requirements (even for expert practitioners).

   A detailed methodology for fine-tuning BERT on annotated aerospace corpora is explained to facilitate the reproducibility of this work and support research in this domain.

3. **Methodology for standardizing aerospace requirements using LMs**: Aerospace requirements were standardized using the LMs developed as a part of this dissertation along with an off-the-shelf text chunking LM.

   (a) **Creation of requirements table**: The requirements table contains columns populated by outputs obtained from the LMs, namely aeroBERT-Classifier, and aeroBERT-NER and presents the requirements along with their properties in a tabular format. This will aid in the creation of model-based

(e.g., SysML) requirement objects by extracting relevant phrases (system names, resources, quantities, etc.) from free-form NL requirements in an automated way.

(b) **Creation of requirements boilerplate**: Various boilerplates were identified for different types of requirements based on the linguistic patterns identified by the LMs.

8.3 Limitations and Recommendations for Future Work

Limitations of this work and avenues for future research directions are discussed below.

8.3.1 aeroBERT-NER

This study focused on identifying five specific types of named entities (SYS, RES, VAL, DATETIME, and ORG) and demonstrated the effectiveness of the proposed methodology. The approach and resulting model, aeroBERT-NER, exhibit generalizability. To build on this work, it would be valuable to train or fine-tune language models that can identify additional types of named entities, which would aid in standardizing requirements further. For instance, named entities related to a system's functional attribute (FUNC) or the performance conditions (COND) under which a system must operate could be of particular interest.

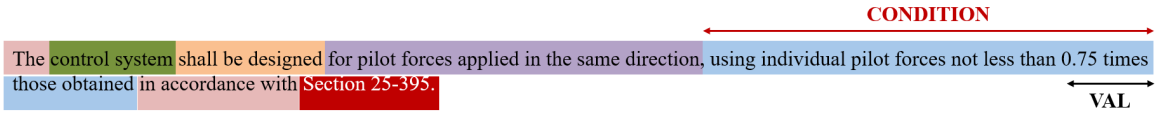


Figure 8.1: Example showing overlapping named entities (VAL and COND) which is an avenue for furthering this work.

This work only considered non-overlapping named entities. However, it was noted that there may be instances where two named entities overlap, particularly if entities

like FUNC and COND are included. As shown in the example in Figure 8.1, the named entities COND and VAL overlap. Thus, another area for future research is to develop a method for identifying overlapping aerospace named entities that will be helpful for the standardization of requirements or for converting information present in NL requirements into data objects. However, another way to achieve this same functionality might be by employing a text chunking along with a NER model, as demonstrated by this work.

This dissertation focused on fine-tuning a pre-trained BERT language model for NER tailored to the aerospace domain. Hence, training an LM on aerospace text from scratch, fine-tuning it for the NER task (using the annotated dataset `archanatikayatray/aeroBERT-NER`), and comparing its performance to that of aeroBERT-NER for NER would be interesting to explore.

8.3.2 aeroBERT-Classifier

When developing the requirements classification language model aeroBERT-Classifier, only three types of requirements were included because there were insufficient examples of other types of requirements in Parts 23 and 25 of Title 14 CFR (which mainly contain certification requirements). As a result, it may be worthwhile to incorporate additional types of requirements from various different sources to enhance the overall applicability of aeroBERT-Classifier.

Small datasets are frequently utilized in the field of requirements engineering because of the scarcity of larger datasets and the expertise needed for annotation. Consequently, aeroBERT-Classifier was fine-tuned on a small annotated aerospace requirements dataset. Nonetheless, exploring the possibility of using additional requirements of each category to train or fine-tune a language model could be a valuable avenue for research. In addition, it would be worthwhile to consider the feasibility of assigning multiple labels to requirements that encompass aspects of more than one

requirement type.

An interesting area for further investigation would be to compare the performance of aeroBERT-Classifier with a LM that is trained from scratch on aerospace requirements and then fine-tuned for the requirements classification task using the annotated dataset `archanatikayatray/aeroBERT-classification`.

8.3.3 Standardization of requirements

This dissertation proposed two methods for standardizing aerospace requirements: creating a requirements table with information extracted from natural language (NL) requirements and identifying boilerplate templates for different requirement types. These approaches were made possible by pre-trained language models fine-tuned on annotated aerospace corpora, resulting in generalizable models. However, the task requires the use of three different LMs working in conjunction to achieve the desired outcomes.

It could be worth exploring the creation of a data pipeline that incorporates different language models to automatically or semi-automatically translate natural language requirements into system models. Nonetheless, this method is likely to involve multiple rounds of refinement and necessitate input from MBSE practitioners.

Exploring the use of generative LMs like T5, the GPT family, etc., may prove beneficial in rewriting freeform NL requirements. These models could be trained on a dataset containing NL requirements and their corresponding rewritten versions that adhere to industry standards. With this training, the model may be capable of generating well-written requirements based on the input NL requirements. Although untested, this approach presents an interesting research direction to explore.

Appendices

APPENDIX A

DATASETS

A.0.1 Accessing the open-source aerospace NER dataset

NER dataset URL: https://huggingface.co/datasets/archanatikayatray/aeroBERT-NER

The following code can be used to bring the annotated NER aerospace corpus into a Python environment. The corpus will be stored in a pandas DataFrame called "dataset". Both `datasets` and `pandas` libraries need to be installed to run the following code chunk successfully.

```
from datasets import load_dataset
import pandas as pd

dataset = load_dataset("archanatikayatray/aeroBERT-NER")

#Converting the dataset into a pandas DataFrame
dataset = pd.DataFrame(dataset["train"]["text"])
dataset = dataset[0].str.split('*', expand = True)

#Getting the headers from the first row
header = dataset.iloc[0]

#Excluding the first row since it contains the headers
dataset = dataset[1:]

#Assigning the header to the DataFrame
dataset.columns = header

#Viewing the last 10 rows of the annotated dataset
dataset.tail(10)
```

A.0.2 Accessing the open-source aerospace requirements classification dataset

Classification dataset URL: https://huggingface.co/datasets/archanatikayatray/aeroBERT-classification

The following code can be used to bring the annotated aerospace requirements corpus into a Python environment. The corpus will be stored in a pandas DataFrame called "dataset". Both `datasets` and `pandas` libraries need to be installed to run the following code chunk successfully.

```
from datasets import load_dataset
import pandas as pd

dataset = load_dataset("archanatikayatray/aeroBERT-classification")

#Converting the dataset into a pandas DataFrame
dataset = pd.DataFrame(dataset["train"]["text"])
dataset = dataset[0].str.split('*', expand = True)

#Getting the headers from the first row
header = dataset.iloc[0]

#Excluding the first row since it contains the headers
dataset = dataset[1:]

#Assigning the header to the DataFrame
dataset.columns = header

#Viewing the last 10 rows of the annotated dataset
dataset.tail(10)
```

APPENDIX B

TEST SET FOR IDENTIFICATION OF NAMED ENTITIES

Table B.1: Test set containing 20 aerospace requirements

Serial No.	Aerospace requirements
1	Flight deck controls must be installed to allow accomplishment of all the tasks required to safely perform the equipment's intended function, and information must be provided to the flightcrew that is necessary to accomplish the defined tasks.
2	Operationally-relevant behavior of the installed equipment must be predictable and unambiguous.
3	A free air temperature indicator or an air-temperature indicator which provides indications that are convertible to free-air temperature must be installed so that the instrument is visible from each pilot station.
4	A clock displaying hours, minutes, and seconds with a sweep-second pointer or digital presentation must be installed so that the instrument is visible from each pilot station.
5	A direction indicator (nonstabilized magnetic compass) must be installed so that the instrument is visible from each pilot station.
6	A machmeter is required at each pilot station for airplanes with compressibility limitations not otherwise indicated to the pilot by the airspeed indicating system required under paragraph (b)(1) of this section.
7	An indicator for the fuel strainer or filter required by Section 25-997 to indicate the occurrence of contamination of the strainer or filter before it reaches the capacity established in accordance with Section 25-997(d) is required.
8	A warning means for the oil strainer or filter required by Section 25-1019, if it has no bypass, to warn the pilot of the occurrence of contamination of the strainer or filter screen before it reaches the capacity established in accordance with Section 25-1019(a)(2) is required.
9	Each electrical and electronic system that performs a function, for which failure would prevent the continued safe flight and landing of the airplane, must be designed and installed so that the function is not adversely affected during and after the time the airplane is exposed to lightning.

Serial No.	Aerospace requirements
10	Before December 1, 2012, an electrical or electronic system that performs a function whose failure would prevent the continued safe flight and landing of an airplane may be designed and installed without meeting the provisions of paragraph (a) provided the system has previously been shown to comply with special conditions for HIRF, prescribed under Section 21-16, issued before December 1, 2007.
11	Each flight, navigation, and powerplant instrument for use by any pilot must be plainly visible to him from his station with the minimum practicable deviation from his normal position and line of vision when he is looking forward along the flight path.
12	The flight instruments required by Section 25-1303 must be grouped on the instrument panel and centered as nearly as practicable about the vertical plane of the pilot's forward vision.
13	Each airspeed indicating instrument must be approved and must be calibrated to indicate true airspeed (at sea level with a standard atmosphere) with a minimum practicable instrument calibration error when the corresponding pitot and static pressures are applied.
14	Each pressure altimeter must be approved and must be calibrated to indicate pressure altitude in a standard atmosphere, with a minimum practicable calibration error when the corresponding static pressures are applied.
15	If a flight instrument pitot heating system is installed, an indication system must be provided to indicate to the flight crew when that pitot heating system is not operating.
16	Each magnetic direction indicator must be installed so that its accuracy is not excessively affected by the airplane's vibration or magnetic fields.
17	The effects of a failure of the system to disengage the autopilot or autothrust functions when manually commanded by the pilot must be assessed in accordance with the requirements of Section 25-1309.
18	Under normal conditions, the disengagement of any automatic control function of a flight guidance system may not cause a transient response of the airplane's flight path any greater than a minor transient.
19	Each powerplant and auxiliary power unit instrument line must meet the requirements of Sections 25-993 and 25-1183.
20	Each powerplant and auxiliary power unit instrument that utilizes flammable fluids must be installed and located so that the escape of fluid would not create a hazard.

APPENDIX C
DEFINITIONS AND NLP CONCEPTS

- **Lemma**: The base form of a word (Figure C.1)

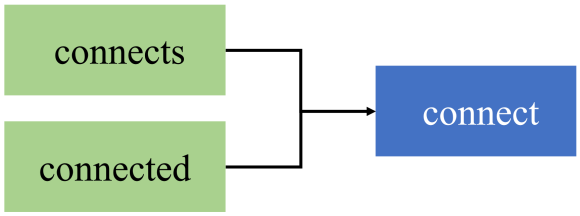


Figure C.1: Lemmatization

- **Dependency Parsing**: Analyzing grammatical structure of a sentence, establishing relationships between the "head" words and the words which modify those heads [133] (Figure C.2)

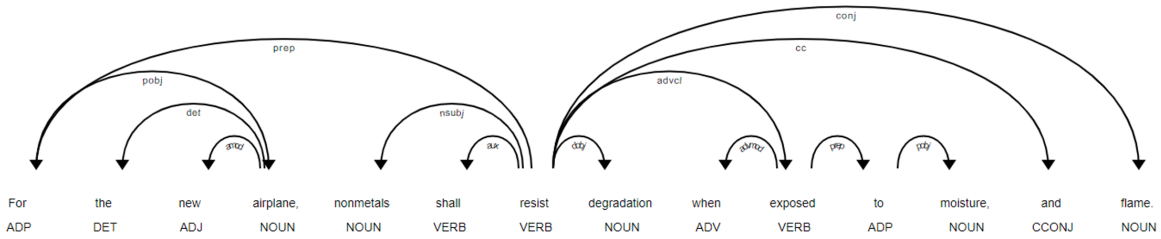


Figure C.2: Dependency Parsing

- **Corpus**: Collection of texts, for example, all the text available on Wikipedia on which language models are trained
- **Word sense disambiguation**: Selecting the meaning of a word with multiple meanings based on context via semantic analysis (Figure C.3)
- **Uncased vs Cased text**: For BERT LM, the text has to be all cased/uncased before being tokenized (Figure C.4)

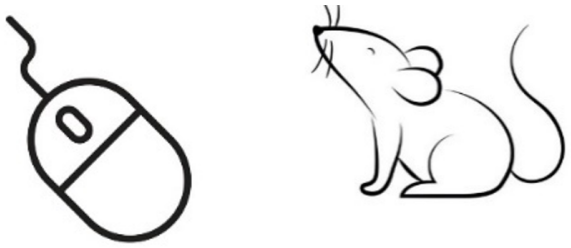


Figure C.3: Word sense disambiguation - A computer mouse vs. a mouse (rodent)

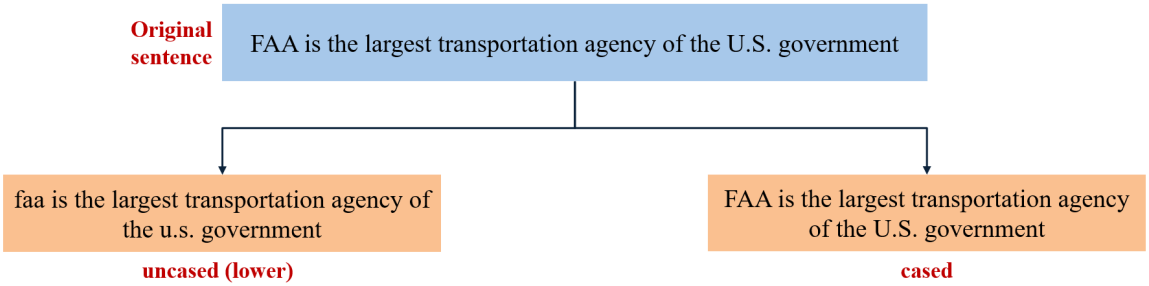


Figure C.4: Cased and uncased text for BERT language model

- **Named-Entity Recognition (NER)**: Involves recognizing named entities such as people, places, organizations, currencies, etc. (Figure C.5)

Figure C.5: NER example

- **Tokenization**: Separating text into smaller units (tokens) such as words, characters, sub-words (Figure C.6)

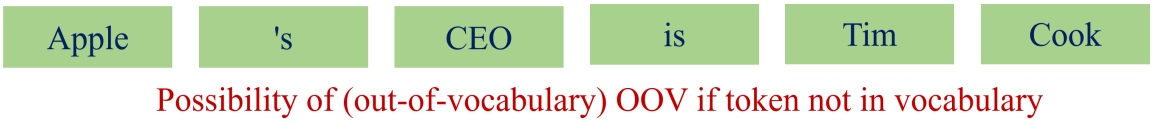


Figure C.6: Tokenization example

- **WordPiece Tokenizer**: A type of tokenizer used by SOTA LMs such as BERT (Figure C.7)

  If a word is not present in the vocabulary, then it is broken down into sub-words/characters that are present in the vocabulary.

embedding = em + ##bed + ##ding

Figure C.7: WordPiece Tokenizer

- **Stop words**: Words that contribute very little to the overall meaning of a sentence and hence can be removed depending on the use case (Example: the, a, an, etc.) (Figure C.8)

The quick brown fox jumps over the lazy dog.

Figure C.8: Stop words

- **Semantic analysis**: Interpreting the meaning of a sentence based on the words in the sequence
- **Syntactic analysis**: Checks the meaningfulness of a sentence by considering the rules of grammar
- **Information retrieval**: Retrieving the most appropriate information from a body of text based on a given query (Example: Google Search)
- **Word Embeddings**: Representing words as vectors, words with similar meanings will have similar representation (Example: Word2Vec, GloVe, etc.) [134] (Figure C.9, Figure C.10)

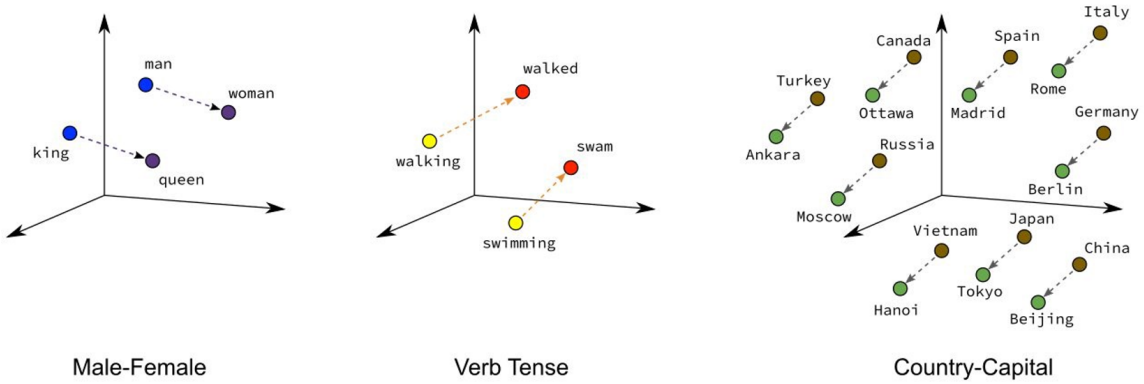


Figure C.9: Word Embeddings in vector space [134]

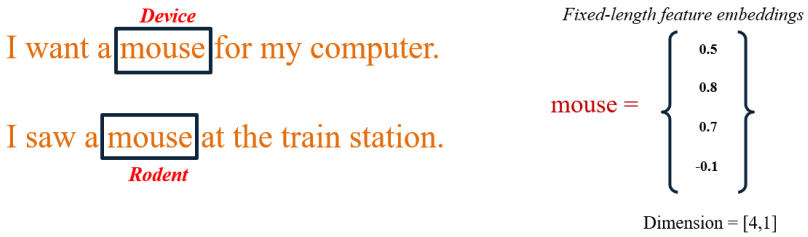


Figure C.10: Word Embeddings Word2Vec

Word2Vec will encode both the "mouse" (Figure C.3) as the same vector irrespective of the context, which is not helpful. Hence, we will look into BERT embeddings that do not have this pitfall.

- **BERT embeddings**: Word embeddings that are dynamically informed by the words around them [38], [135] (Figure C.11, Figure C.12)

Figure C.11: BERT Embeddings

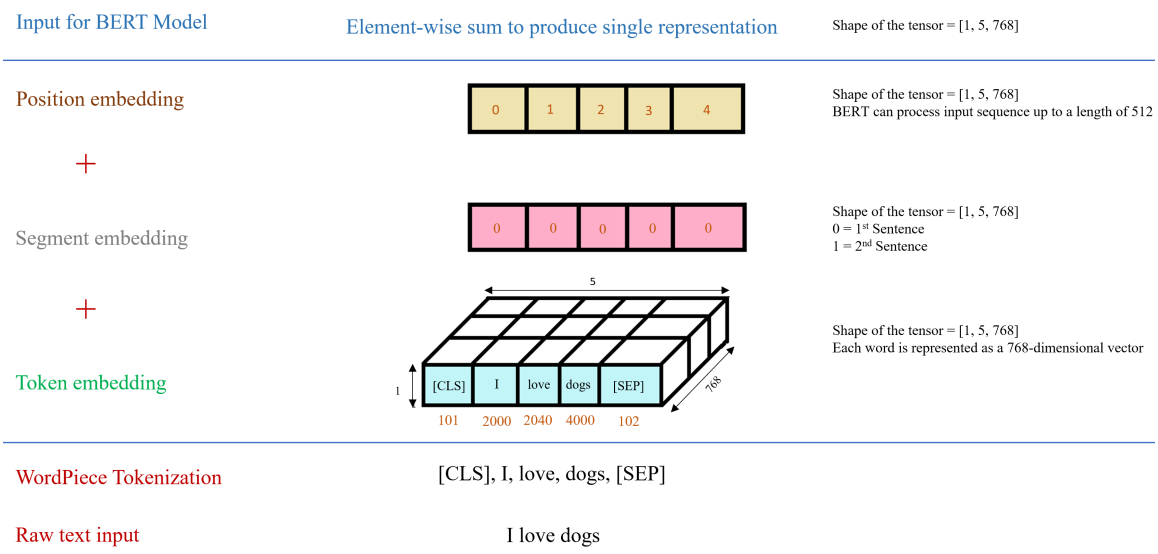


Figure C.12: BERT Embeddings deep-dive [38], [135]

- **Large Language Models (LLMs)**: LLMs are advanced natural language processing (NLP) models that use deep learning algorithms to analyze and understand human language. They are pre-trained on vast amounts of text

data (Wikipedia, BookCorpus, etc.) and have the ability to generate coherent and contextually appropriate language responses, making them ideal for a wide range of NLP tasks such as language translation, question-answering, text summarization, and sentiment analysis. Examples of LLMs include GPT-3, BERT, and XLNet.

- **Transformers**: Transformers are neural network architectures that rely on attention mechanisms to process input data. This allows the model to attend to different parts of the input sequence with varying weights, allowing it to capture long-range dependencies and context more effectively. A transformer consists of two main components or blocks: the encoder and the decoder [37] (Figure C.13).

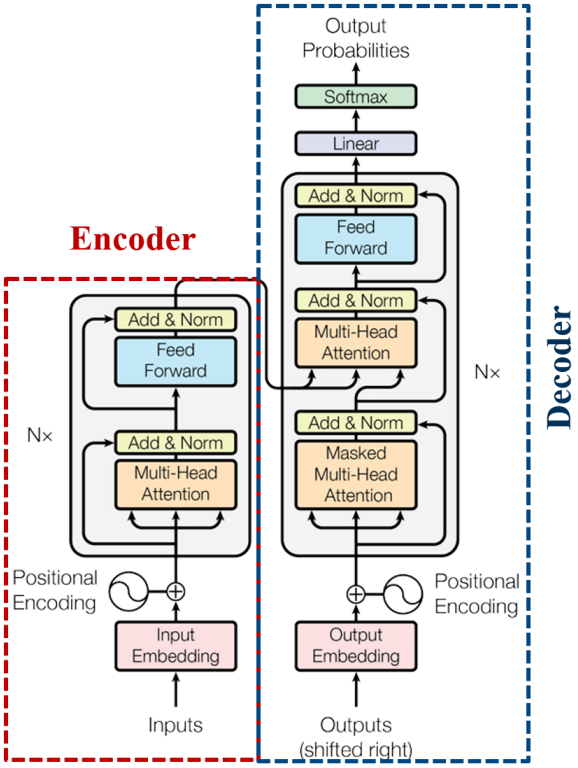


Figure C.13: Transformer Architecture showing the encoder and decoder blocks [37]

- **Encoder Block**: The encoder block is responsible for processing the input sequence and extracting its features. It consists of a stack of identical layers,

with each layer containing two sub-layers: a self-attention mechanism and a feedforward neural network. The self-attention mechanism allows the encoder to attend to different parts of the input sequence, while the feedforward network processes the attended information [37] (Figure C.13). BERT_{BASE} contains 12 encoder blocks and BERT_{LARGE} contains 24 encoder layers [38].

- **Decoder Block**: The decoder block is responsible for generating the output sequence based on the extracted features from the encoder. It also consists of a stack of identical layers, each with two sub-layers: a masked self-attention mechanism that allows the decoder to attend to its own previously generated outputs and a cross-attention mechanism that allows it to attend to the encoder's output [37] (Figure C.13).

- **Self-attention (scaled dot product attention)**: In a typical self-attention mechanism, the input sequence is first transformed into three different vectors: Query (Q), Key (K), and Value (V). The query vector represents the current position in the sequence, while the key and value vectors represent all positions in the sequence. The self-attention mechanism then computes a weight for each position in the sequence, based on how relevant it is to the current position. This weight is computed by taking the dot product between the query and key vectors and scaling the result by the square root of the dimensionality of the key vector. The resulting weights are then used to compute a weighted sum of the value vectors, which is the output of the self-attention mechanism [37]. The tokenized input matrix (X) depicted in Figure C.14 displays each row as an embedding for a token, consisting of the token embedding, segment embedding, and position embedding. This specific example used for illustration purposes only comprises three tokens, with each token possessing four dimensions. However, for the BERT model, every token has 768 dimensions.

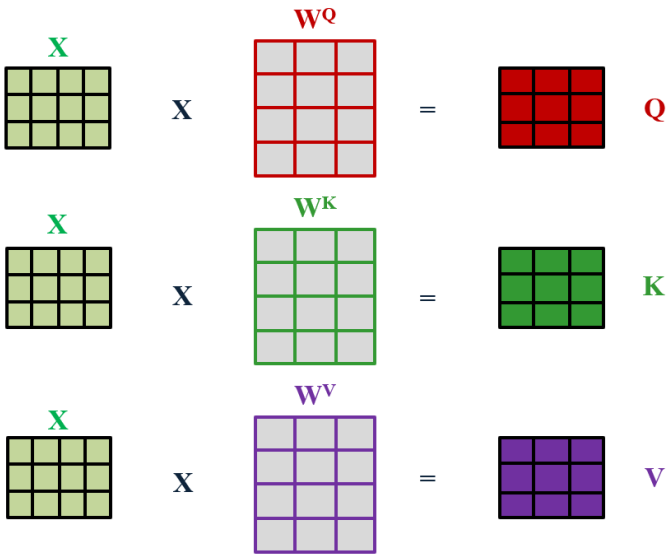


Figure C.14: Query, Key, and Value matrices [37]

In order to map the tokenized input (X) onto query space, key space, and value space, three weight matrices - W^Q, W^K, and W^V - were multiplied with the input matrix, resulting in Q, K, and V, respectively.

Figure C.15 illustrates an instance that demonstrates the calculation of attention scores for the word "like" in the sentence "*I like dogs*" using Query (Q) and Key (K) vectors. The diagram displays the significance of "*like*" concerning each of the other words in the sequence. The Key vector (K) for "*like*" attracts more attention, with an attention score of 0.87, indicating that the Q and K vectors were close, resulting in more attention between the two tokens. In this instance, the sum of attention scores does not equal 1. Additionally, it is worth noting that the Key and Query vectors exist in distinct vector spaces. It is crucial to keep in mind that these figures are provided solely for the purpose of illustration.

In this example, the Value matrix (V) has a dimension of 3 ($d_k = 3$), and its square root will be utilized to normalize the attention score via Equation 3.1. Lastly, the attention scores obtained are multiplied with the Value matrix (V)

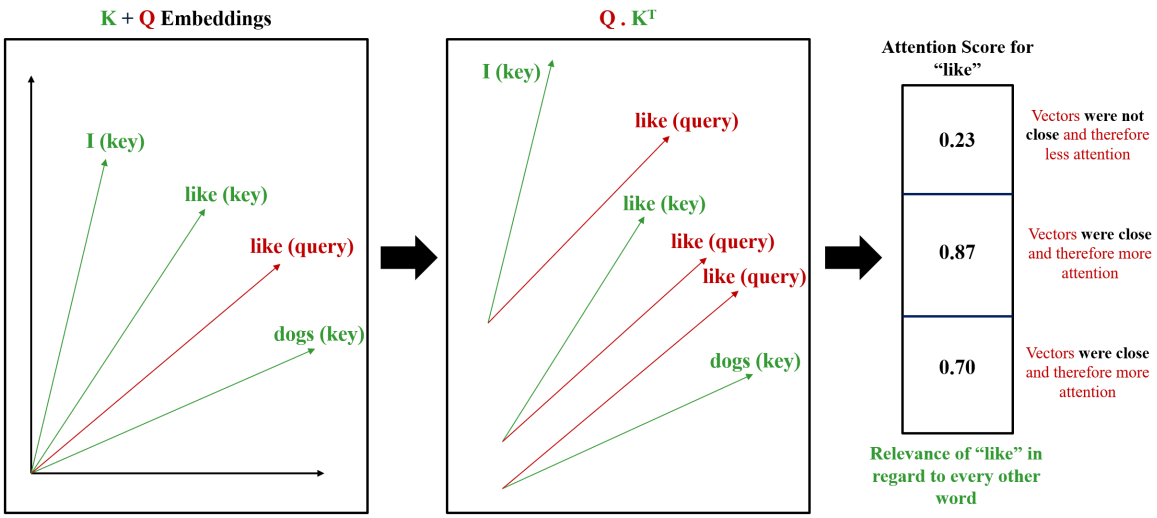


Figure C.15: Example showing Query (Q) and Key (K) vectors and attention score calculation for the word "*like*" in the sentence "*I like dogs*". Key and Query live in different vector spaces. These figures are for demonstration purposes only.

to generate the "*context-ful*" embedding for the word "*like*". This process is repeated for each word/token present in the sequence.

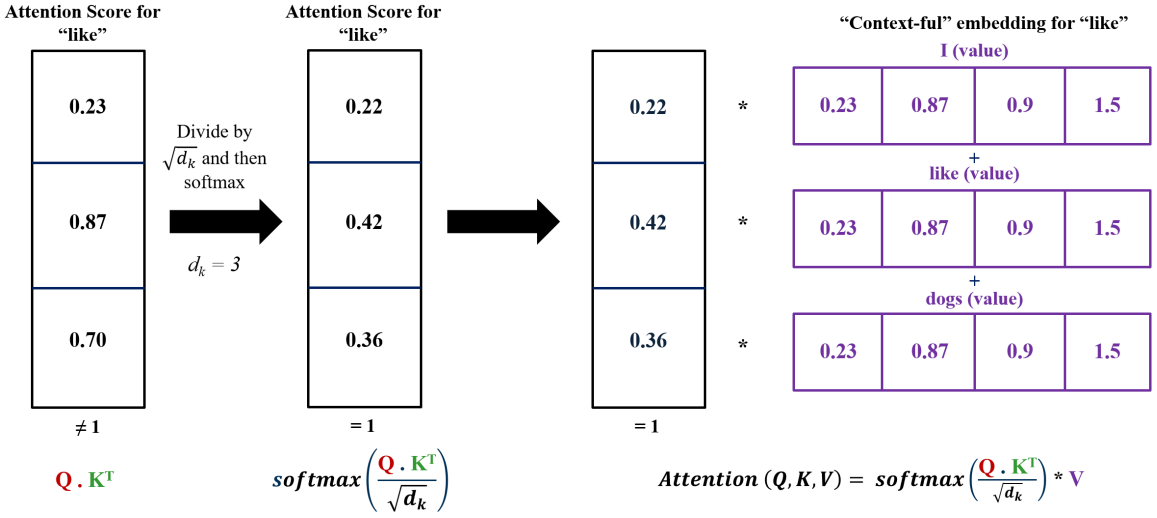


Figure C.16: Example showing Query (Q) and Key (K) vectors and attention score calculation for the word "*like*" in the sentence "*I like dogs*". The equations for the calculation of attention are shown. Key and Query live in different vector spaces. These figures are for demonstration purposes only.

- **Multi-headed attention**: Multi-headed attention is a mechanism used in transformer-based models, such as the Transformer and BERT, to improve the

performance of self-attention by allowing the model to attend to different parts of the input sequence simultaneously. In multi-headed attention, the input sequence is transformed into multiple representations (or "heads") using different sets of learned Query (Q), Key (K), and Value (V) matrices. Each head operates independently, allowing the model to capture different types of relationships between the input sequence elements. The output of each head is then concatenated and transformed into the final output. This approach enhances the model's capacity to attend to different aspects of the input sequence and extract more complex relationships between them (Figure C.17). Multi-headed attention has been shown to be effective in various natural language processing tasks, such as machine translation, text classification, and named entity recognition.

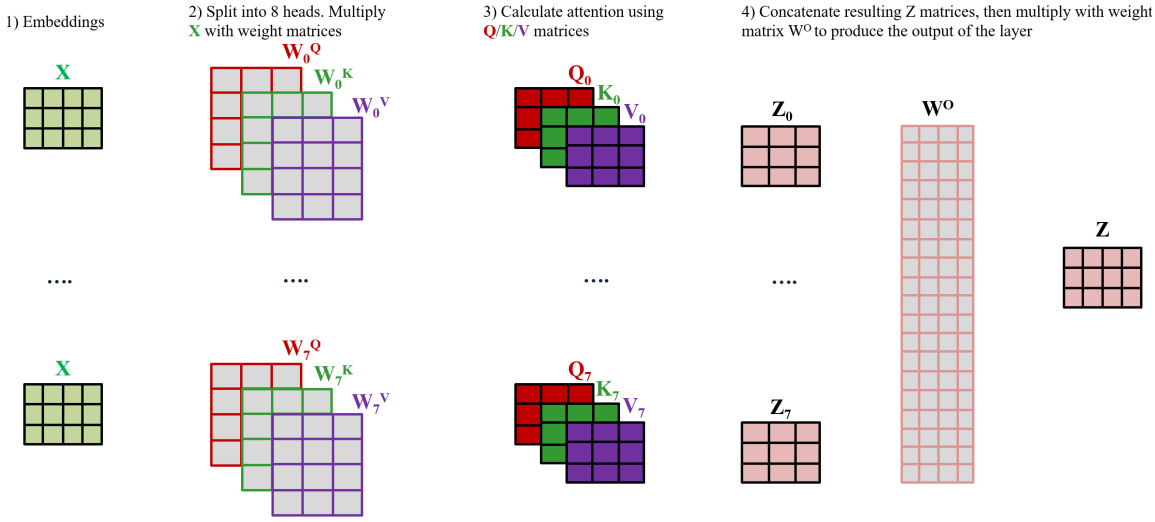


Figure C.17: Matrix multiplication intuition for the calculation of multi-headed attention [104]

REFERENCES

[1] D. P. Schrage, "Technology for rotorcraft affordability through integrated product/process development (ippd)," 1999.

[2] "Enabling technologies for strategic decision-making of advanced design concepts," *AE 6373: Advanced Design Methods 1, Georgia Institute of Technology*,

[3] "Systems engineering fundamentals," *Wayback Machine Defense Acquisition University Press*, 2001.

[4] "Guide to the systems engineering body of knowledge," in (BKCASE Editorial Board), BKCASE Editorial Board. INCOSE, 2020, p. 945.

[5] INCOSE, "Incose infrastructure working group charter," pp. 3–5, (accessed: 01.10.2023).

[6] NASA, "Appendix c: How to write a good requirement," no. 5, pp. 115–119, (accessed: 01.05.2022).

[7] NASA, "2.1 the common technical processes and the se engine," *J. Object Technol.*, vol. 4, no. 1, (accessed: 01.10.2023).

[8] B. Nuseibeh and S. Easterbrook, "Requirements engineering: A roadmap," in *Proceedings of the Conference on The Future of Software Engineering*, ser. ICSE '00, Limerick, Ireland: Association for Computing Machinery, 2000, pp. 35–46, ISBN: 1581132530.

[9] B. Regnell, R. B. Svensson, and K. Wnuk, "Can we beat the complexity of very large-scale requirements engineering?" In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, Springer, 2008, pp. 123–128.

[10] Google, "Natural language — google arts culture," Google, (accessed: 01.10.2022).

[11] F. Dalpiaz, A. Ferrari, X. Franch, and C. Palomares, "Natural language processing for requirements engineering: The best is yet to come," *IEEE software*, vol. 35, no. 5, pp. 115–119, 2018.

[12] M. Luisa, F. Mariangela, and N. I. Pierluigi, "Market research for requirements analysis using linguistic tools," *Requirements Engineering*, vol. 9, no. 1, pp. 40–56, 2004.

[13] T. E. Bell and T. A. Thayer, "Software requirements: Are they really a problem?" In *Proceedings of the 2nd international conference on Software engineering*, 1976, pp. 61–68.

[14] D. Firesmith, "Common requirements problems, their negative consequences, and the industry best practices to help solve them.," *J. Object Technol.*, vol. 6, no. 1, pp. 17–33, 2007.

[15] B. Haskins, J. Stecklein, B. Dick, G. Moroney, R. Lovell, and J. Dabney, "8.4. 2 error cost escalation through the project life cycle," in *INCOSE International Symposium*, Wiley Online Library, vol. 14, 2004, pp. 1723–1737.

[16] IBM. "Overview of doors." (accessed: 01.10.2023). (2023).

[17] J. C. Knight, "Safety critical systems: Challenges and directions," in *Proceedings of the 24th international conference on software engineering*, 2002, pp. 547–550.

[18] P. Vallejo, R. Mazo, C. Jaramillo, and J. M. Medina, "Towards a new template for the specification of requirements in semi-structured natural language," *Journal of Software Engineering Research and Development*, vol. 8, pp. 3–1, 2020.

[19] H. Yang, A. De Roeck, V. Gervasi, A. Willis, and B. Nuseibeh, "Speculative requirements: Automatic detection of uncertainty in natural language requirements," in *2012 20th IEEE International Requirements Engineering Conference (RE)*, IEEE, 2012, pp. 11–20.

[20] A. Fatwanto, "Software requirements specification analysis using natural language processing technique," in *2013 International Conference on QiR*, IEEE, 2013, pp. 105–110.

[21] D. M. Berry *et al.*, "From contract drafting to software specification: Linguistic sources of ambiguity-a handbook version 1.0," 2000.

[22] R. Kuehl and J. S. Hawker, "Requirements analysis-ambiguity," 2009.

[23] A. Arellano, E. Zontek-Carney, and M. A. Austin, "Frameworks for natural language processing of textual requirements," *International Journal On Advances in Systems and Measurements*, vol. 8, pp. 230–240, 2015.

[24] J. Oberg, "Why the mars probe went off course [accident investigation]," *IEEE Spectrum*, vol. 36, no. 12, pp. 34–39, 1999.

[25] A. L. Ramos, J. V. Ferreira, and J. Barceló, "Model-based systems engineering: An emerging approach for modern systems," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 1, pp. 101–111, 2011.

[26] J. A. Estefan *et al.*, "Survey of model-based systems engineering (mbse) methodologies," *Incose MBSE Focus Group*, vol. 25, no. 8, pp. 1–12, 2007.

[27] L. Jacobson and J. R. G. Booch, "The unified modeling language reference manual," 2021.

[28] M. Ballard, R. Peak, S. Cimtalay, and D. N. Mavris, "Bidirectional text-to-model element requirement transformation," *IEEE Aerospace Conference*, pp. 1–14, 2020.

[29] L. Lemazurier, V. Chapurlat, and A. Grossetête, "An mbse approach to pass from requirements to functional architecture," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 7260–7265, 2017.

[30] "Needs, requirements, verification, validation lifecycle manual," in (BKCASE Editorial Board), BKCASE Editorial Board. INCOSE, 2022, p. 457.

[31] L. Wheatcraft, M. Ryan, J. Llorens, and J. Dick, "The need for an information-based approach for requirement development and management," in *INCOSE International Symposium*, Wiley Online Library, vol. 29, 2019, pp. 1140–1157.

[32] A. Ferrari, F. Dell'Orletta, A. Esuli, V. Gervasi, and S. Gnesi, "Natural language requirements processing: A 4d vision.," *IEEE Softw.*, vol. 34, no. 6, pp. 28–35, 2017.

[33] R. J. Abbott and D. Moorhead, "Software requirements and specifications: A survey of needs and languages," *Journal of Systems and Software*, vol. 2, no. 4, pp. 297–316, 1981.

[34] C. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. Bethard, and D. McClosky, "The Stanford CoreNLP natural language processing toolkit," in *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, Baltimore, Maryland: Association for Computational Linguistics, Jun. 2014, pp. 55–60.

[35] "Natural language toolkit." (accessed: 01.10.2023). ().

[36] "Spacy." (accessed: 01.10.2023). ().

[37] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[38] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, *Bert: Pre-training of deep bidirectional transformers for language understanding*, 2019. arXiv: 1810.04805 [cs.CL].

[39] M. Lewis, Y. Liu, N. Goyal, *et al.*, "BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," *CoRR*, vol. abs/1910.13461, 2019. arXiv: 1910.13461.

[40] L. Zhao, W. Alhoshan, A. Ferrari, *et al.*, "Natural language processing for requirements engineering: A systematic mapping study," *ACM Computing Surveys (CSUR)*, vol. 54, no. 3, pp. 1–41, 2021.

[41] Z. Liu, B. Li, J. Wang, and R. Yang, "Requirements engineering for crossover services: Issues, challenges and research directions," *IET Software*, vol. 15, no. 1, pp. 107–125, 2021. eprint: https://ietresearch.onlinelibrary.wiley.com/doi/pdf/10.1049/sfw2.12014.

[42] A. Mavin, P. Wilkinson, S. Teufl, H. Femmer, J. Eckhardt, and J. Mund, "Does goal-oriented requirements engineering achieve its goal?" In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, 2017, pp. 174–183.

[43] U. Eklund, H. Holmström Olsson, and N. J. Strøm, "Industrial challenges of scaling agile in mass-produced embedded systems," in *Agile Methods. Large-Scale Development, Refactoring, Testing, and Estimation*, T. Dingsøyr, N. B. Moe, R. Tonelli, S. Counsell, C. Gencel, and K. Petersen, Eds., Cham: Springer International Publishing, 2014, pp. 30–42, ISBN: 978-3-319-14358-3.

[44] R. Sonbol, G. Rebdawi, and N. Ghneim, "The use of nlp-based text representation techniques to support requirement engineering tasks: A systematic mapping review," *IEEE Access*, vol. PP, pp. 1–1, 2022.

[45] L. Zhao, W. Alhoshan, A. Ferrari, and K. J. Letsholo, *Classification of natural language processing techniques for requirements engineering*, 2022.

[46] F. Gilardi, M. Alizadeh, and M. Kubli, "Chatgpt outperforms crowd-workers for text-annotation tasks," *arXiv preprint arXiv:2303.15056*, 2023.

[47] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[48] W. Alhoshan, L. Zhao, A. Ferrari, and K. J. Letsholo, "A zero-shot learning approach to classifying requirements: A preliminary study," in *Requirements Engineering: Foundation for Software Quality*, V. Gervasi and A. Vogelsang, Eds., Cham: Springer International Publishing, 2022, pp. 52–59, ISBN: 978-3-030-98464-9.

[49] J. Cleland-Huang, S. Mazrouee, H. Liguo, and D. Port. "Nfr." (Mar. 2007).

[50] E. D. Liddy, "Natural language processing," 2001.

[51] A. Singh, K. Ramasubramanian, and S. Shivam, "Natural language processing, understanding, and generation," in *Building an Enterprise Chatbot*, Springer, 2019, pp. 71–192.

[52] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: Smaller, faster, cheaper and lighter," *ArXiv*, vol. abs/1910.01108, 2019.

[53] Y. Goldberg, "Neural network methods for natural language processing," *Synthesis lectures on human language technologies*, vol. 10, no. 1, pp. 1–309, 2017.

[54] D. Jurafsky and J. H. Martin, "Speech and language processing (draft)," 2021.

[55] T. R. Niesler and P. C. Woodland, "A variable-length category-based n-gram language model," in *1996 IEEE International Conference on Acoustics, Speech, and Signal Processing Conference Proceedings*, IEEE, vol. 1, 1996, pp. 164–167.

[56] Y. Bengio, R. Ducharme, and P. Vincent, "A neural probabilistic language model," in *Advances in Neural Information Processing Systems*, T. Leen, T. Dietterich, and V. Tresp, Eds., vol. 13, MIT Press, 2000.

[57] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[58] A. Graves, "Generating sequences with recurrent neural networks," Aug. 4, 2013. arXiv: 1308.0850v5 [cs.NE].

[59] K. Cho, B. van Merrienboer, Ç. Gülçehre, *et al.*, "Learning phrase representations using rnn encoder–decoder for statistical machine translation," in *EMNLP*, 2014.

[60] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015.

[61] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *Advances in neural information processing systems*, vol. 27, 2014.

[62] F. Morin and Y. Bengio, "Hierarchical probabilistic neural network language model," in *International workshop on artificial intelligence and statistics*, PMLR, 2005, pp. 246–252.

[63] T. Brown, B. Mann, N. Ryder, *et al.*, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33, Curran Associates, Inc., 2020, pp. 1877–1901.

[64] C. Raffel, N. Shazeer, A. Roberts, *et al.*, "Exploring the limits of transfer learning with a unified text-to-text transformer," *Journal of Machine Learning Research*, vol. 21, no. 140, pp. 1–67, 2020.

[65] C. Sun, X. Qiu, Y. Xu, and X. Huang, "How to fine-tune bert for text classification?" In *China national conference on Chinese computational linguistics*, Springer, 2019, pp. 194–206.

[66] J. Alammar. "The illustrated bert, elmo, and co. (how nlp cracked transfer learning)." (2018). (accessed: 02.21.2022).

[67] A. Dima, S. Lukens, M. Hodkiewicz, T. Sexton, and M. P. Brundage, "Adapting natural language processing for technical text," *Applied AI Letters*, vol. 2, no. 3, e33, 2021.

[68] Y. Gu, R. Tinn, H. Cheng, *et al.*, "Domain-specific language model pretraining for biomedical natural language processing," *ACM Transactions on Computing for Healthcare (HEALTH)*, vol. 3, no. 1, pp. 1–23, 2021.

[69] C. Arora, M. Sabetzadeh, L. Briand, and F. Zimmer, "Automated extraction and clustering of requirements glossary terms," *IEEE Transactions on Software Engineering*, vol. 43, no. 10, pp. 918–945, 2016.

[70] Z. Kurtanović and W. Maalej, "Automatically classifying functional and non-functional requirements using supervised machine learning," in *2017 IEEE 25th International Requirements Engineering Conference (RE)*, Ieee, 2017, pp. 490–495.

[71] A. M. Davis, *Software requirements: objects, functions, and states*. Prentice-Hall, Inc., 1993.

[72] G. Kotonya and I. Sommerville, *Requirements engineering: processes and techniques*. John Wiley & Sons, Inc., 1998.

[73] Z. S. H. Abad, O. Karras, P. Ghazi, M. Glinz, G. Ruhe, and K. Schneider, "What works better? a study of classifying requirements," in *2017 IEEE 25th International Requirements Engineering Conference (RE)*, IEEE, 2017, pp. 496–501.

[74] J. D. Palmer, Y. Liang, and L. Want, "Classification as an approach to requirements analysis," *Advances in Classification Research Online*, vol. 1, no. 1, pp. 131–138, 1990.

[75] C. Duan, P. Laurent, J. Cleland-Huang, and C. Kwiatkowski, "Towards automated requirements prioritization and triage," *Requirements engineering*, vol. 14, no. 2, pp. 73–89, 2009.

[76] J. L. Cybulski and K. Reed, "Requirements classification and reuse: Crossing domain boundaries," in *International Conference on Software Reuse*, Springer, 2000, pp. 190–210.

[77] J. Cleland-Huang, R. Settimi, X. Zou, and P. Solc, "The detection and classification of non-functional requirements with application to early aspects," in *14th IEEE International Requirements Engineering Conference (RE'06)*, IEEE, 2006, pp. 39–48.

[78] J. Cleland-Huang, R. Settimi, X. Zou, and P. Solc, "Automated classification of non-functional requirements," *Requirements engineering*, vol. 12, no. 2, pp. 103–120, 2007.

[79] A. Rashwan, O. Ormandjieva, and R. Witte, "Ontology-based classification of non-functional requirements in software specifications: A new corpus and svm-based classifier," in *2013 IEEE 37th Annual Computer Software and Applications Conference*, 2013, pp. 381–386.

[80] J. Winkler and A. Vogelsang, "Automatic classification of requirements based on convolutional neural networks," in *2016 IEEE 24th International Requirements Engineering Conference Workshops (REW)*, IEEE, 2016, pp. 39–45.

[81] R. Jindal, R. Malhotra, and A. Jain, "Automated classification of security requirements," in *2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, IEEE, 2016, pp. 2027–2033.

[82] M. Binkhonain and L. Zhao, "A review of machine learning algorithms for identification and classification of non-functional requirements," *Expert Systems with Applications: X*, vol. 1, p. 100 001, 2019.

[83] T. Hey, J. Keim, A. Koziolek, and W. F. Tichy, "Norbert: Transfer learning for requirements classification," in *2020 IEEE 28th International Requirements Engineering Conference (RE)*, IEEE, 2020, pp. 169–179.

[84] "Zero-shot learning in modern nlp." (accessed: 01.10.2023). ().

[85] W. Yin, J. Hay, and D. Roth, "Benchmarking zero-shot text classification: Datasets, evaluation and entailment approach," *CoRR*, vol. abs/1909.00161, 2019. arXiv: 1909.00161.

[86] A. Williams, N. Nangia, and S. Bowman, "A broad-coverage challenge corpus for sentence understanding through inference," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, New Orleans, Louisiana: Association for Computational Linguistics, 2018, pp. 1112–1122.

[87] C. Arora, M. Sabetzadeh, L. Briand, and F. Zimmer, "Automated checking of conformance to requirements templates using natural language processing," *IEEE transactions on Software Engineering*, vol. 41, no. 10, pp. 944–968, 2015.

[88] C. Arora, M. Sabetzadeh, L. C. Briand, and F. Zimmer, "Requirement boilerplates: Transition from manually-enforced to automatically-verifiable natural language patterns," in *2014 IEEE 4th International Workshop on Requirements Patterns (RePa)*, IEEE, 2014, pp. 1–8.

[89] A. Rajan and T. Wahl, *CESAR: Cost-efficient methods and processes for safety-relevant embedded systems*. Springer, 2013.

[90] A. Ruiz, M. Sabetzadeh, P. Panaroni, *et al.*, "Challenges for an open and evolutionary approach to safety assurance and certification of safety-critical systems," in *2011 First International Workshop on Software Certification*, IEEE, 2011, pp. 1–6.

[91] J. M. C. De Gea, J. Nicolás, J. L. F. Alemán, A. Toval, C. Ebert, and A. Vizcaıno, "Requirements engineering tools: Capabilities, survey and assessment," *Information and Software Technology*, vol. 54, no. 10, pp. 1142–1157, 2012.

[92] C. Rupp, *Requirements-Engineering und-Management: professionelle, iterative Anforderungsanalyse für die Praxis*. Hanser Verlag, 2007.

[93] N. Ibrahim, W. M. N. Wan Kadir, and S. Deris, "Documenting requirements specifications using natural language requirements boilerplates," in *2014 8th. Malaysian Software Engineering Conference (MySEC)*, 2014, pp. 19–24.

[94] A. Mavin, P. Wilkinson, A. Harwood, and M. Novak, "Easy approach to requirements syntax (ears)," in *2009 17th IEEE International Requirements Engineering Conference*, IEEE, 2009, pp. 317–322.

[95] C. Arora, M. Sabetzadeh, L. Briand, F. Zimmer, and R. Gnaga, "Automatic checking of conformance to requirement boilerplates via text chunking: An industrial case study," in *2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, IEEE, 2013, pp. 35–44.

[96] T. Rose, N. Haddock, and R. Tucker, "The effects of corpus size and homogeneity on language model quality," 1997.

[97] I. Beltagy, K. Lo, and A. Cohan, "Scibert: A pretrained language model for scientific text," *arXiv preprint arXiv:1903.10676*, 2019.

[98] A. Ferrari, G. O. Spagnolo, and S. Gnesi, "Towards a dataset for natural language requirements processing.," in *REFSQ Workshops*, 2017.

[99] D. Araci, "Finbert: Financial sentiment analysis with pre-trained language models," *arXiv preprint arXiv:1908.10063*, 2019.

[100] J. Lee, W. Yoon, S. Kim, *et al.*, "Biobert: A pre-trained biomedical language representation model for biomedical text mining," *Bioinformatics*, vol. 36, no. 4, pp. 1234–1240, 2020.

[101] E. Alsentzer, J. R. Murphy, W. Boag, *et al.*, "Publicly available clinical bert embeddings," *arXiv preprint arXiv:1904.03323*, 2019.

[102] J.-S. Lee and J. Hsiang, "Patentbert: Patent classification with fine-tuning a pre-trained bert model," *arXiv preprint arXiv:1906.02124*, 2019.

[103] O. Sharir, B. Peleg, and Y. Shoham, "The cost of training nlp models: A concise overview," *arXiv preprint arXiv:2004.08900*, 2020.

[104] J. Alammar. "The illustrated transformer." (2018). (accessed: 02.21.2022).

[105] R. Aggarwal, *Bi-lstm*, 2019, (accessed: 02.21.2022).

[106] K. Clark, U. Khandelwal, O. Levy, and C. D. Manning, "What does BERT look at? an analysis of bert's attention," *CoRR*, vol. abs/1906.04341, 2019. arXiv: 1906.04341.

[107] E. F. Tjong Kim Sang and F. De Meulder, "Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition," in *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, 2003, pp. 142–147.

[108] S. Quiniou, P. Cellier, T. Charnois, and D. Legallois, "What about sequential data mining techniques to identify linguistic patterns for stylistics?" In *International Conference on Intelligent Text Processing and Computational Linguistics*, Springer, 2012, pp. 166–177.

[109] M. Warnier and A. Condamines, "Improving requirement boilerplates using sequential pattern mining," in *Europhras 2017*, 2017.

[110] C. C. Aggarwal and C. Zhai, "A survey of text classification algorithms," in *Mining text data*, Springer, 2012, pp. 163–222.

[111] K. Kowsari, K. Jafari Meimandi, M. Heidarysafa, S. Mendu, L. Barnes, and D. Brown, "Text classification algorithms: A survey," *Information*, vol. 10, no. 4, p. 150, 2019.

[112] N. Sanchez-Pi, L. Martı, and A. C. B. Garcia, "Text classification techniques in oil industry applications," in *International Joint Conference SOCO'13-CISIS'13-ICEUTE'13*, Springer, 2014, pp. 211–220.

[113] "The eight parts of speech." (2019). (accessed: 02.21.2022).

[114] A. Akbik, D. Blythe, and R. Vollgraf, "Contextual string embeddings for sequence labeling," in *COLING 2018, 27th International Conference on Computational Linguistics*, 2018, pp. 1638–1649.

[115] D. Jurafsky. "Speech and language processing – chapter 8 slides." (). (accessed: 02.21.2022).

[116] K. Kravari, C. Antoniou, and N. Bassiliades, "Towards a requirements engineering framework based on semantics," PCI 2020, pp. 72–76, 2021.

[117] G. Fanmuy, A. Fraga, and J. Llorens, "Requirements verification in the industry," *Proceedings of the 2nd International Conference on Complex Systems Design and Management, CSDM 2011*, pp. 145–160, Jan. 2011.

[118] Y. Wautelet, S. Heng, M. Kolp, and I. Mirbel, "Unifying and extending user story models," M. Jarke, J. Mylopoulos, C. Quix, *et al.*, Eds., pp. 211–225, 2014.

[119] FAA, *Overview — title 14 of the code of federal regulations (14 cfr)*, 2013, (accessed: 02.21.2022).

[120] A. T. Ray, O. J. Pinon-Fischer, D. N. Mavris, R. T. White, and B. F. Cole, "Aerobert-ner: Named-entity recognition for aerospace requirements engineering using bert," in *AIAA SCITECH 2023 Forum*.

[121] "Fundamentals of systems engineering: Requirements definition." (accessed: 01.10.2023). ().

[122] L. S. Wheatcraft. "Everything you wanted to know about interfaces, but were afraid to ask." (2017). (accessed: 01.10.2023).

[123] J. Spacey. "11 examples of quality requirements." (2017). (accessed: 10.19.2022).

[124] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in arXiv, 2017.

[125] R. Weischedel, M. Palmer, M. Marcus, *et al.* "Ontonotes release 5.0." (accessed: 01.10.2023). (2013).

[126] A. Tikayat Ray, B. F. Cole, O. J. Pinon Fischer, R. T. White, and D. N. Mavris, "Aerobert-classifier: Classification of aerospace requirements using bert," *Aerospace*, vol. 10, no. 3, 2023.

[127] "Syntax." (accessed: 02.21.2023). ().

[128] "Requirement table." (accessed: 02.10.2023). ().

[129] "Modeling requirements with sysml." (accessed: 02.10.2023). ().

[130] M. Riesener, C. Dölle, A. Becker, S. Gorbatcheva, E. Rebentisch, and G. Schuh, "Application of natural language processing for systematic requirement management in model-based systems engineering," *INCOSE International Symposium*, vol. 31, no. 1, pp. 806–815, 2021. eprint: https://incose.onlinelibrary.wiley.com/doi/pdf/10.1002/j.2334-5837.2021.00871.x.

[131] X. Wang, Y. Jiang, N. Bach, *et al.*, "Automated concatenation of embeddings for structured prediction," in *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, Online: Association for Computational Linguistics, Aug. 2021, pp. 2643–2660.

[132] J. Pennington, R. Socher, and C. Manning, "GloVe: Global vectors for word representation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1532–1543.

[133] "Dependency parsing." (The Stanford Natural Language Processing Group). (accessed: 02.21.2022).

[134] "Translating to a lower-dimensional space." (Google). (accessed: 02.21.2022).

[135] C. McCormick and N. Ryan. "Bert word embeddings tutorial." (2019). (accessed: 02.21.2022).