Efficacy of Deep Neural Networks in Natural Language Processing for Classifying Requirements by Origin and Functionality: An Application of BERT in System Requirements

by

Jesse Mullis

(Under the Direction of Beshoy Morkos)

Abstract

Given the foundational role of system requirements in design projects, designers can benefit from classifying, comparing, and observing connections between requirements. Manually undertaking these processes, however, can be laborious and time-consuming. Previous studies have employed Bidirectional Encoder Representations from Transformers (BERT), a natural language processing model, to automatically analyze written requirements. Yet, it remains unclear whether BERT can capture the nuances that differentiate requirements. This work evaluates BERT's performance on two requirement classification tasks executed on five system design documents. First, a BERT model is fine-tuned to classify requirements according to their originating project. A separate BERT model is then fine-tuned to classify each requirement as either functional or nonfunctional. The former model receives a Matthews correlation coefficient (MCC) of 0.95, while the latter receives an MCC of 0.82. This work then explores the application of BERT's requirement representations to identify similar requirements and predict requirement change.

Index words: [Design Automation, Requirements Management, Natural Language Processing, BERT]

Efficacy of Deep Neural Networks in Natural Language Processing for Classifying Requirements by Origin and Functionality: An Application of BERT in System Requirements

by

Jesse Mullis

B.S., University of Georgia, 2020

A Thesis Submitted to the Graduate Faculty of the
University of Georgia in Partial Fulfillment of the Requirements for the Degree

Master of Science

Athens, Georgia

2022

Efficacy of Deep Neural Networks in Natural Language Processing for Classifying Requirements by Origin and Functionality: An Application of BERT in System Requirements

by

Jesse Mullis

Acknowledgments

"Four things do not come back: the spoken word, the sped arrow, the past life, and the neglected opportunity."

— Ted Chiang, *The Merchant and the Alchemist's Gate*

I would like to begin by thanking all of those who have made this opportunity possible for me. First and foremost, I offer a tremendous thanks to my advisor, Dr. Beshoy Morkos, for welcoming me into his lab. His guidance has been critical not only to this research, but also to my growth as a person over the past two years. I would also like to thank Dr. Scott Ferguson for graciously offering his expertise and Dr. Ben Davis, who seems to have a limitless supply of insight and has served as a model of excellence throughout my time at the University of Georgia. Thanks are also due to my labmates, who have always offered help in times of need and have surely become lifelong friends. Finally, I would like to thank my remaining friends, family, church, and, in particular, my fiancee, Jana Demian, for their continual love and support.

Contents

List of Figures

List of Tables

List of Abbreviations

ARCPP	Automated Requirement Change Propagation Prediction
BERT	Bidirectional Encoder Representations from Transformers
CBOW	Continuous Bag-of-words
CPM	Change Prediction Method
DSM	Design Structure Matrix
ECN	Engineering Change Notification
Faiss	Facebook Artificial Intelligence Similarity Search
FC	Functional Classification
FR	Functional Requirement
HOQ	House of Quality
LSTM	Long Short-term Memory
MCC	Matthews Correlation Coefficient
NFR	Nonfunctional Requirement
NLP	Natural Language Processing
PDC	Parent Document Classification
RNN	Recurrent Neural Network
SB	Sentence-BERT
SKA	Square Kilometer Array
t-SNE	T-distributed Stochastic Neighbor Embeddings

Chapter 1

Introduction

Engineering designers rely on requirements to document, understand, and fulfill stakeholder needs. Following the general framework of content-based recommendation systems [1], research in the field of requirement engineering seeks to create systems for requirement retrieval [2, 3]. The challenge in finding relevant design information is emphasized by the tremendous amount of digitized data available across domains [4]. Consequently, automated procedures are preferred to laborious manual searches. Requirement engineers express a specific interest in the retrieval of requirements for their use in applications including requirement tracing [3], linking [5], reuse [6], and change prediction [7]. Addressing these requirements management challenges is crucial to project success, particularly when developing complex systems. This work seeks to investigate the efficacy of Bidirectional Encoder Representations from Transformers (BERT), a deep neural network, for representing requirements in a format suitable for automated analysis. The resulting representations have potential applications in requirements reuse and change prediction.

The reuse of requirements is especially necessary when designing new product variations that are unlikely to change the solution principle and instead only adapt the embodiment to new requirements and constraints. In this design method, known as adaptive design, the functional structure can be modified by variation, addition, or omission since the general structure is well-known [8]. An adaptive design approach, commonly employed for complex, custom systems [9], necessitates identification of similar requirements across different projects. For example, the Mars 2020 Perseverance Rover has many system requirements that overlap with its predecessor, the Mars Science Laboratory Curiosity Rover. With each project containing at minimum hundreds of requirements, designers may strain to identify similarities across the requirement corpus, leading to missed opportunities for reuse.

Though requirement elicitation occurs early in the design process, requirements often do not remain permanent throughout design projects. Requirement change propagation is a common phenomena that occurs when an alteration to one requirement necessitates the subsequent change of other requirements. Such change propagation can lead to unexpected project delays and expenses. Consequently, designers and stakeholders stand to benefit greatly from tools that can predict requirement change propagation ahead of time [10]. Previous research has shown that change propagation can be effectively predicted by the semantic and syntactic similarity of written requirements, where requirements' similarity is proportional to the likelihood of change propagating from one requirement to another [11]. However, previously proposed methods for measuring requirement similarity are computationally expensive and fail to capture a sufficiently detailed language representation. Advances in natural language processing (NLP) have led to the introduction of models, like BERT, that could improve requirements management practices.

Namely, BERT can project textual requirements into a continuous vector space conducive to computer analysis. The resulting vector representations are known as language embeddings [12]. Because of its utilization of transfer learning, BERT comes prepackaged with a highly developed understanding of natural language that can be fine-tuned to specific requirements management tasks. Previous studies have applied BERT to requirements [13, 14, 15, 16], but it has yet to be shown whether BERT can reliably recognize variations in requirements that occur between industry projects. It is also unknown whether BERT can distinguish between types of requirements, i.e., functional requirements (FRs) and nonfunctional requirements (NFRs), within industry projects. Therefore, this work looks to answer the following research questions:

- **RQ1**: Can BERT's requirement embeddings differentiate across requirements documents?
- **RQ2**: Can BERT's requirement embeddings differentiate between FRs and NFRs?

- **RQ3**: Do embeddings from BERT fine-tuned on requirement classification tasks yield better results in requirements management applications than general-language embeddings?

Figure 1.1 displays a map of this research and indicates where each research question is addressed in the overall process. The research questions are investigated using a dataset of 1,303 requirements sourced from five mechanical design documents. Three of the documents come from private industry and detail the design of various manufacturing equipments, while the remaining two documents are publicly available and detail the design of subsystems for the Square Kilometer Array (SKA). To answer RQ1, a BERT model is fine-tuned for a "parent document classification (PDC)" task, where its objective is to classify requirements according to the project they came from. Then, to answer RQ2, a separate BERT model is fine-tuned for a "functional classification (FC)" task with the objective of classifying requirements as either FRs or NFRs.

After evaluating each model's performance, potential applications of the resulting requirement embeddings are explored; embeddings from the PDC BERT model are used to identify similar requirements both at the document and individual requirement level, while embeddings from the FC BERT model are applied to predict requirement change in two of the design projects. To answer RQ3, the results are compared to those produced by a Sentence-BERT (SB) model trained to create sentence embeddings for general language. Beyond testing BERT's ability to sufficiently represent requirements, this work determines whether fine-tuning on requirement classification yields higher quality requirement embeddings than a general-language SB model.

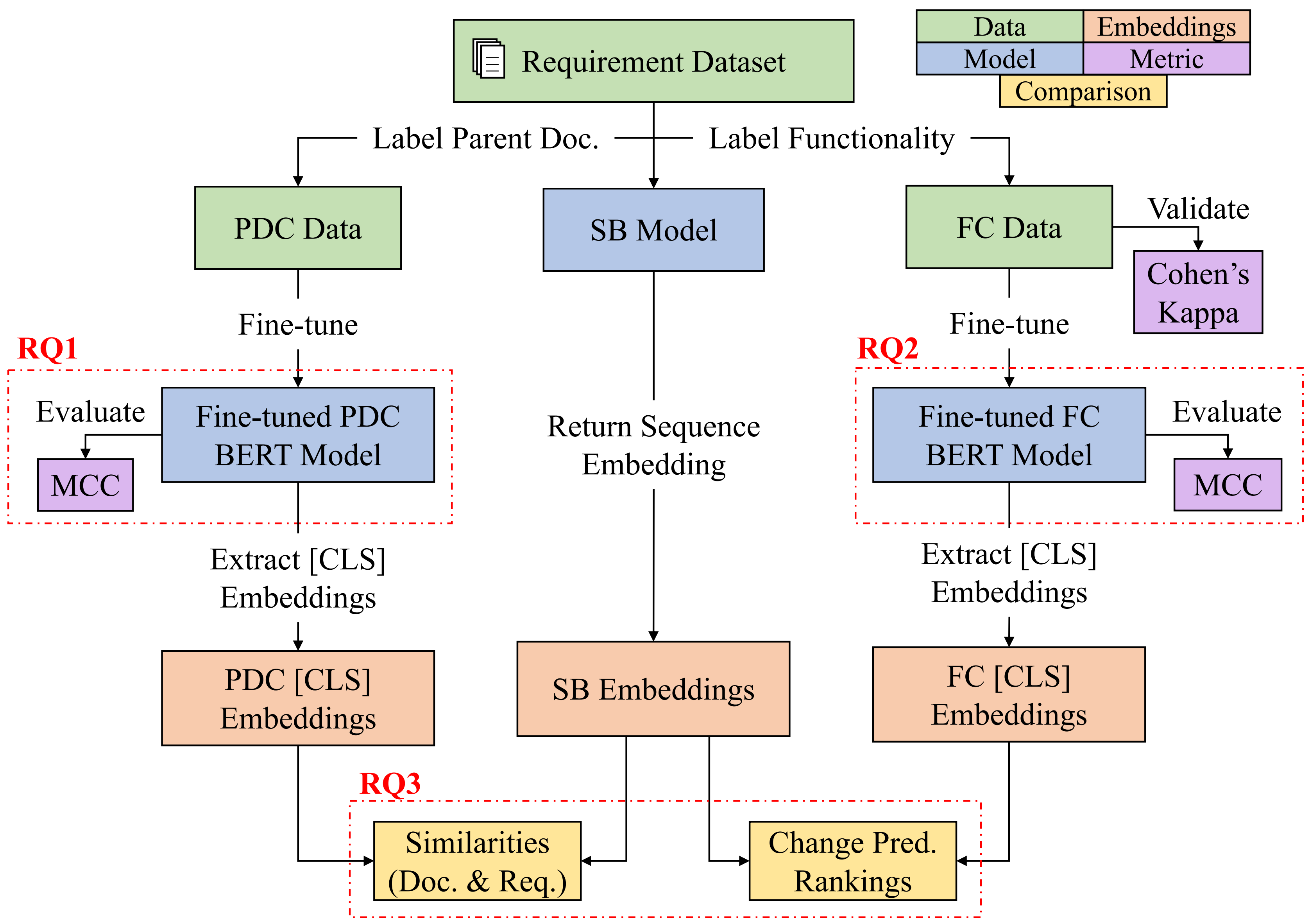


Figure 1.1: Research Logic Map.

Chapter 2

Background

2.1 Overview

To begin, this work should be placed in its greater context as the next piece in a progression of engineering requirements research from the MODE^{2}L (Manufacturing Optimization, Design, and Engineering Education Lab) Group and its predecessors. This lineage stems from the creation of the Automated Requirement Change Propagation Prediction (ARCPP) tool, which predicts requirement change by examining relationships derived from higher-order design structure matrices (DSMs) [17]. The initial work was motivated by a lack of formal reasoning in requirements management tools [18] and was developed through multiple industry-sponsored projects with applications in management [19] and manufacturing [20]. Prior to the introduction of ARCPP, engineering change prediction had only been considered at the component, rather than the requirement, level. A subsequent study reveals that requirement relationships can be extracted from text using part-of-speech tagging and a bag-of-words method [11], opening the door for further language-based studies of requirements; later investigation suggests that requirement change is best predicted through the physical (noun), rather than the functional (verb), domain [21, 22, 23]. An alternative change prediction approach relying on complex network centrality metrics was found to yield results similar to those generated by the ARCPP tool [24]. The need for change prediction tools was reaffirmed by studies that link continuous requirement evolution with success in capstone design projects

[25, 26, 27]. Though requirement change was established as unavoidable and necessary, further research was required to determine its origins [28]. One study helps designers to prepare for change by employing machine learning to characterize requirement change volatility, which describes the behavior a requirement exhibits due to an initial change [29]. Additional work supports earlier findings that NFRs behave distinct from FRs throughout the design process [30]. To further investigate ideal requirement groupings, another study experiments with dividing requirements into sets based on topics extracted through Latent Dirichlet Allocation [31]. This current work intends to extend those mentioned by evaluating the ability of NLP's deep neural network models (here, BERT serves as an ambassador for all such models) to represent and automatically analyze engineering requirements for applications such as change prediction. Should these models prove as capable at requirements management tasks as they are with typical NLP tasks, then exciting new opportunities become available for the engineering requirement research community. For example, automated requirement text analysis could support current research that explores the use of FRs and NFRs to identify excess in modular systems [32, 33, 34].

The following background sections examine the fields of engineering requirements and NLP as pertains to the creation and applications of requirement embeddings generated by BERT models fine-tuned on requirement classification tasks. Section 2.2 focuses on design requirements, their management, and existing research on requirement classification, requirement similarity, and requirement change prediction. Then, Section 2.3 reviews research that has led to BERT, a state-of-the-art NLP model. Finally, Section 2.4 provides a summary of the findings and existing gaps.

2.2 Design Requirements

In engineering settings, requirements are the purpose, goals, constraints, and criteria associated with a design project [11]. Requirements serve as the primary mode of communication and documentation of stakeholder needs and play a role in each stage of the design process, from project conception to completion. Unsurprisingly, they have been found to greatly impact a project's overall success [8]. Designers have consequently developed methods, such as those detailed in seminal design books [8, 35], to systematically elicit, compose, and manage requirements.

Preferably, designers form requirements as testable, unambiguous statements that bound a design space for allowable solutions [36]. Requirements should not, however, be so particular as to result in overspecifications that introduce complexity and unnecessary features [37]. Throughout a design project, requirements undergo the processes of elicitation, specification, validation, and verification [38]. Requirements elicitation describes the process in which designers consult stakeholders to identify apparent and latent stakeholder needs. During the specification phase, designers refine the appropriate requirements into explicit, testable statements. In the validation phase, designers and stakeholders review requirements for "consistency, completeness, and correctness" [39]. The involvement of both designers and stakeholders in the validation process ensures mutual agreement prior to committing further resources to product design. Finally, requirements verification entails proving, through methods such as inspection, modeling, or expert review, that requirements have been fulfilled [40]. Effective use of requirements in each stage of the design process necessitates that requirements follow a particular structure.

Though the particulars of requirement formats do, and indeed should [41, 42], vary from industry to industry and project to project, requirements typically follow the same basic pattern. Designers most often express requirements as imperative statements containing the verbs "shall," "should," or "will." Appendix C in the NASA Systems Engineering Handbook [43] includes a checklist of recommendations for writing a good requirement. The checklist suggests that designers use "shall" for explicit requirements, "should" for goals, and "will" for facts or declarations of purpose. Following are some more of the handbook's recommendations:

- Requirements should be stated positively (e.g., use "shall" instead of "shall not") with correct grammar and spelling.
- Requirements should convey one thought with a single subject and predicate.
- Indefinite pronouns, ambiguous words, and unverifiable terms should be avoided.

- Consistent terminology should be used throughout the document.

These recommendations describe a grammar for effective communication through requirements. Other guidelines, such as the "Guide for Writing Requirements" by the International Council on Systems Engineering [44] and "Simplified Technical English" by the AeroSpace and Defense Industries Association of Europe [45], establish this grammar in further detail. Commercial software packages, like QRA's QVscribe[1] and IBM's Requirement Quality Assistant[2], offer automatic assessments of requirement quality based on the previously mentioned industry-standard requirements guides. The existence of these guides and the associated software packages suggests the following:

1. Requirements follow a set of rules that distinguish them from generic written text.
2. Properly formed requirements are instrumental to the later requirements management process.

Based on these observations, a logical conclusion is that the representations used for requirement analysis should not be general language representations, but should instead be tailored specifically to requirements. It remains unclear whether modern NLP models, like BERT, can capture the nuances that distinguish requirements between and within engineering projects. Prior to reviewing representations in Section 2.3, this section will further explore requirements management, including the two challenges this work seeks to address: requirement similarity and requirement change prediction.

2.2.1 Requirements Management

Proper requirements management is critical to a project's successful completion [46]. Here, "requirements management" refers to activities that support the eliciting, documenting, validating, tracing, analyzing, and verifying of requirements. The House of Quality (HOQ) is a popular and well-established design framework made popular by Hauser and Clausing's article in the Harvard Business Review [47]. The HOQ provides several useful insights for requirements management. First, the HOQ defines relationships between stakeholders' desired attributes and the elicited technical requirements. Establishing

[1]https://qracorp.com/qvscribe
[2]https://www.ibm.com/products/requirements-quality-assistant

these relationships validates whether requirements do indeed describe a solution space agreeable with the stakeholder desires. A HOQ may indicate requirements that satisfy one or multiple desired attributes, while simultaneously detracting from the fulfillment of other stakeholder desires; such requirements may require additional attention, modification, or even elimination. Similarly, the HOQ indicates inter-requirement relationships, where the fulfillment of some requirements may coincide with, or contest, the fulfillment of others. Lastly, the HOQ aids in requirements verification by documenting target values for each technical requirement.

While the HOQ remains a relevant framework, many designers today rely on software packages such as IBM DOORS[3] or Jama[4] to manage requirements. These packages can represent requirement dependencies through a hierarchy while also keeping track of "metadata," which is any pertinent data aside from the requirement text itself. Metadata may include things like an author, date, reason of origin, or method of verification, and allows designers to trace all aspects of a requirement from project conception to completion. A downside of proprietary softwares when compared to the methods proposed in this paper, however, is their cost and inability to adapt source code to meet project-specific needs. More importantly, these commercial software packages do not supply a means of automatically identifying opportunities for requirements reuse; Jama, for example, offers a way to reuse content from previous projects, but it remains the designer's responsibility to determine which requirements should be reused. QVscribe provides requirement similarity analysis that could prove useful in this regard if it were able to evaluate requirement similarity across projects. Instead, the current implementation limits similarity assessments to requirements in the same project. It can also be difficult to use commercial software to track the impact of requirement changes. After a change occurs to one requirement, the software examines dependencies and flags related requirements for review. The dependencies, however, must be manually input by the designer. Establishing dependencies is not only time consuming, but also subjective; in some scenarios change may propagate between requirements that the designer initially considered to be independent. To

[3]https://www.ibm.com/products/requirements-management
[4]https://www.jamasoftware.com

fully address requirements reuse and change, designers require tools in addition to the existing commercial software.

2.2.2 Classifying Requirements

Though the HOQ and requirements management software emphasize the need to examine requirements relationships, they do not categorize these relationships themselves. Rather, they rely on designers to sort requirements into groups, with the broadest two groups being FRs and NFR. Distinguishing between FRs and NFR is a crucial step in requirements management. Not only do FRs appear earlier in the elicitation process than their nonfunctional counterparts [8], but they also pose different challenges and have varying effects on project success [25]. This paper follows Shankar's definition of FRs as "what a product must do, be able to perform, or should do" [30]. While there is broad agreement on this definition of FRs, there is no such consensus for the definition of NFRs [48]. Chung et al. attempt to define NFRs through a list of "-ilities" (e.g., compatibility, reliability) [49]. Other descriptors not ending in "ility," such as legal, operational, and security, can also be grouped under NFRs [50]. In order to maintain a binary, generalizable classification of requirements, this work considers NFRs to simply be any requirement that is not a FR.

The process of manually sorting requirements is time-consuming and labor-intensive, especially in complex systems with hundreds, if not thousands, of requirements. Several works attempt to automatically classify requirements with machine learning. Kurtanović and Maajel use a support vector machine model to classify requirements as FRs or NFRs using lexical features [50]. They use the PROMISE NFR dataset [51] containing 625 student-generated requirements mixed with user requirements extracted from online reviews. They achieve a precision and recall of around 92% for classifying FRs and NFRs and precision of 93% and recall of 90% when further classifying NRFs into usability, security, operational, and performance requirements. Similar works apply other machine learning approaches to the PROMISE dataset, including convolutional neural network (CNN) [52, 53] and BERT [13]. Of the studies mentioned, a fine-tuned BERT model yields the best results, with an F1 score of 92%. A limitation to these studies, however, is that student-generated requirements from the PROMISE NFR dataset may not represent industry-standard requirements. Additionally, the dataset and the research built upon it are geared toward software requirements and may not generalize to mechanical design requirements.

Akay and Sang-Gook bridge this gap between automated requirement classification and mechanical design; they use the generative pre-trained transformer model [54] to create a synthetic dataset of 91,259 mechanical design requirements labeled as FRs or design parameters [55]. They then use BERT to create requirement embeddings that are fed to a support vector machine for classification. Though an impressive accuracy of 99.1% is achieved on a test set, the model is not validated on an authentic set of requirements. Even though the synthetic requirements are mechanical in nature, they may not reflect the nuances of industry requirements. Therefore, a study is needed to verify the ability of advanced NLP models, like BERT, to classify a diverse set of requirements gathered from industry. It should also be noted that requirements can be split into other useful groups based on system architecture or domains of expertise (e.g., electrical, chemical, mechanical, software, etc.). In the design of large, complex systems, grouping requirements in these fashions helps to allocate requirements and resources to the appropriate team. Researchers continue to search for optimal groupings; Chen et al., for example, explore requirement clusters autonomously generated from latent topics [31]. A workflow for fine-tuning BERT to classify industrial requirements into FRs and NFRs could later be applied to other types of requirement classification. Additionally, fine-tuning BERT for requirement classification tasks may yield requirement representations that could prove useful in other tasks, like identifying similar requirements and predicting requirement change.

2.2.3 Identifying Similar Requirements

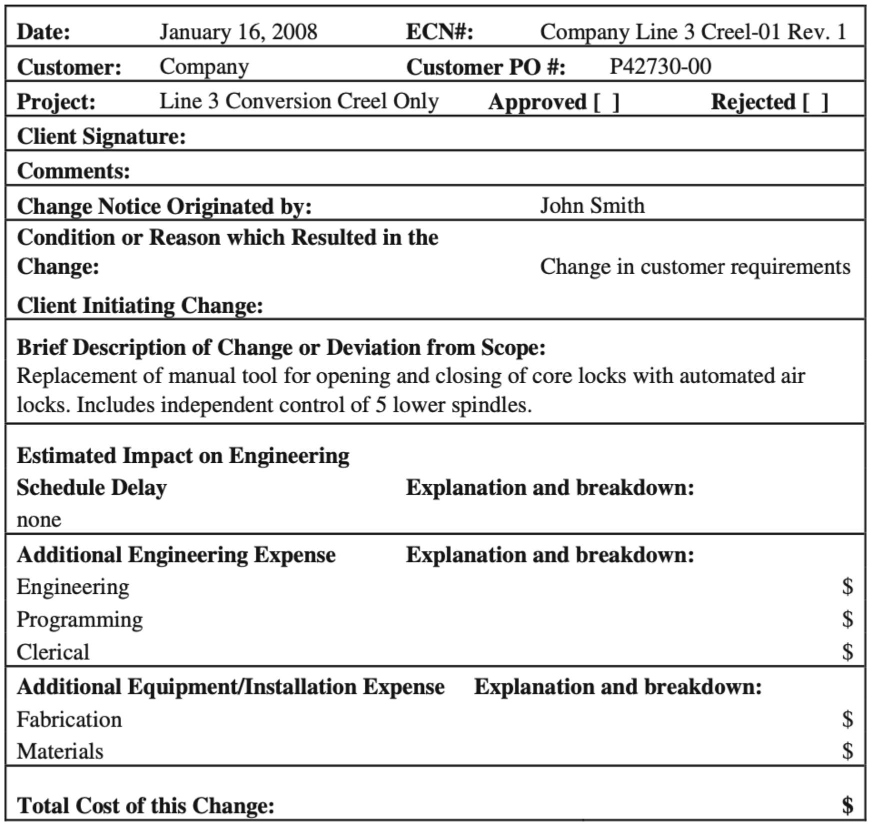
In the design of mechanical systems, adaptive design, where "one keeps to known and established solution principles and adapts the embodiment to changed requirements," and variant design, where "the sizes and arrangements of parts and assemblies are varied within the limits set by previously designed product structures," both necessitate the need to identify similar requirements [8]. As systems become increasingly complex, there is greater incentive to build upon previous solutions through adaptation and reuse. Existing research has recognized the need for utilizing previous design data, but largely ignores requirements reuse. For example, Schubert et al. addressed the need for Computer-Aided Design and Failure Mode and Effects Analysis models that can be more efficiently reused, but do not address how the corresponding requirements could be identified across projects [56]. While Yang et al. proposed a model for creating variant designs that satisfy changing customer requirements, they provide no evaluation of the similarity between existing and emerging requirements [57]. Though there exists a need to identify similar requirements within adaptive and variant design of mechanical products, the majority of research investigating requirements similarity comes from the field of software engineering.

Within software engineering, NLP approaches are applied to compute requirements' semantic similarity. Consequently, advances in requirements similarity research have largely coincided with advances in NLP. The framework proposed by Mihany et al. computes a similarity percentage for requirements documents based on shared words [6]. Other works explore the use of term frequency - inverse document frequency as well as latent semantic indexing [58, 59]. Rajpathak et al. create a novel semantic similarity model that examines multi-phrase terms to identify "High," "Low," or "No Link" relationships between requirements [5]. More recent works employ neural network models to create links between software requirements. For example, Guo et al. applied two recurrent neural network (RNN) architectures, long short-term memory (LSTM) [60] and gated recurrent unit [61], to replace previously state-of-the-art tracing methods [62]. The T-BERT framework then outperforms the RNN approach by applying a BERT model that has undergone both intermediate training and fine-tuning to associate natural language artifacts with corresponding programming language artifacts [15]. Abbas et al. evaluated the underlying assumption that semantic similarity relates to software similarity; they compared several different language models' evaluation of semantic similarity and found BERT's results to have the highest correlation with actual software similarity [16]. The BERT model used, though, is an SB model, trained to produce general sentence embeddings. Previous work has shown that requirements documents differ from generic text in terms of structure and terminology [63], so a model trained to represent requirements specifically is preferred. The success of an SB model motivates this work's investigation of a BERT model fine-tuned specifically to distinguish between requirements.

2.2.4 Requirement Change Prediction

In the context of mechanical design, an engineering change is defined as "an alteration made to parts, drawings or software that have already been released in the product design process" [64]. Figure 2.1 illustrates an example of an engineering change notification (ECN), which is the document used to communicate and track the details of a change, such as its initiated date, cause, and the affected requirements [11]. While changes are inevitable and necessary to enhance designs or allow them to address new needs, they often come at the expense of delays and additional costs. Design projects are especially hampered by engineering change propagations, where implementing design changes results in the need for subsequent, unanticipated changes [65]. Designers have responded to this challenge by developing methods of predicting engineering change propagation.

Clarkson et al. [66] introduced the Change Prediction Method (CPM) for anticipating change propagation using DSMs. CPM relies on designers to manually define DSMs that indicate the likelihood and potential impact of an initial change propagating to other components. The model then computes a risk of change propagation based on the likelihood and impact DSMs. A case study applying this model to product design at Westland Helicopters indicated its ability to predict changes not evident from direct dependencies between components, even in the design of a complex mechanical system. A downside of CPM, however, is that designers must manually create the likelihood and impact DSMs, a process which requires extensive time and labor and relies on subjective designer input. Lee and Hong [67] remodel CPM as a Bayesian network capable of learning propagation probabilities from a prior distribution and empirical data, but their model still relies on experts to specify the network structure and estimate the initial prior distribution. A further limitation of these approaches is that they operate on the component level, while changes may occur prior to component specification.



Date: January 16, 2008		**ECN#:** Company Line 3 Creel-01 Rev. 1	
Customer: Company		**Customer PO #:** P42730-00	
Project: Line 3 Conversion Creel Only		**Approved [ ]**	**Rejected [ ]**
Client Signature:			
Comments:			
Change Notice Originated by:		John Smith	
Condition or Reason which Resulted in the Change:		Change in customer requirements	
Client Initiating Change:			
Brief Description of Change or Deviation from Scope: Replacement of manual tool for opening and closing of core locks with automated air locks. Includes independent control of 5 lower spindles.			
Estimated Impact on Engineering			
Schedule Delay	**Explanation and breakdown:**		
none			
Additional Engineering Expense	**Explanation and breakdown:**		
Engineering			$
Programming			$
Clerical			$
Additional Equipment/Installation Expense	**Explanation and breakdown:**		
Fabrication			$
Materials			$
Total Cost of this Change:			**$**

Figure 2.1: Example Engineering Change Notification [11].

In contrast, work by Morkos et al. implements change prediction at the requirement level using higher-order DSMs [17]. Higher-order DSMs track requirement relations beyond direct relationships. For example, if requirement "A" has a direct relationship with requirement "B," and requirement "B" has a direct relationship with requirement "C," then requirement "A" has a second order relationship with requirement "C". Identifying relationships of order three and higher follows the same logic. The reliance on higher-order DSMs allows the model to predict change propagations that could not be identified by first order relationships alone, which are the most obvious to human designers. A follow-up study compared three methods of identifying requirement relationships: manual, linguistic, and neural-network-based [11]. Though the presented linguistic approach provides the greatest accuracy in change prediction, it requires a labor-intensive process for tagging the desired parts of speech. Further work suggests that the accuracy of change prediction depends on whether requirements are linked through functional or non-functional relationships [22]. A model able to automatically represent semantic relationships while also distinguishing functional and nonfunctional relationships, such as BERT fine-tuned on the FC task, may present an opportunity for improved requirement change prediction.

2.3 Language Representations in Vector Space

Written language is perhaps humanity's most powerful tool for communication. People can use it to precisely represent, or at the very least describe in detail, just about anything. Since written text's inception, people have acquired massive collections of text ranging from scientific documents in university libraries to 280 character opinions on Twitter. All of this data presents an opportunity for analysis. Useful information such as an article's topic composition or fluctuations in public opinion can be extracted directly from text, and while humans can reliably discern a text's topic or an author's sentiment, it can be labor-intensive to manually perform this analysis for each document in a large corpus. Modern computers and machine learning algorithms seem to have the potential to perform this analysis autonomously and quickly (relative to a human counterpart). An issue arises, however, in representing language in ways that can be interpreted by a computer while still retaining the same information as the original text.

Statistical language models based on N-grams were, up until the mid-2000s, the dominant models used for NLP tasks. These models determine the probability of a sequence of words based on the conditional probability of each word occurring after the previous N-1 words; bigrams (N=2) only take the previous word into account, while trigrams (N=3) take the two previous words into account. Conditional probabilities are computed by counting the number of occurrences of the N length sequence within a corpus and then normalizing it by the number of other sequences in the corpus that begin with the same N-1 words. Though models generally perform better with higher N, diminishing returns for increased computational complexity usually limit N to 5. The relative simplicity of the N-gram approach allows for models to be trained on large corpora: in 2006 Google released a set of 5-grams obtained from a corpus with over 1 trillion words [68]. Even with models of this scale, however, N-grams are limited in that new sequences of words are constantly generated. They offer no way to apply generalizations from a training corpus to a test set containing novel sequences. In short, N-grams fail to sufficiently capture meaning from language.

Another approach is to represent text as vectors of real numbers. These vectors are known as language embeddings, and are conducive to computer analysis. Most commonly, embeddings are produced at the level of tokens, which are defined as "an instance of a sequence of characters in some document that are grouped together as a useful semantic unit for processing" [69]. It is also possible to create embeddings for entire sentences [70], paragraphs [71], or documents [72]. A simple method for creating word embeddings is to use one-hot encodings, where a vector has an entry corresponding to each word in a vocabulary. Each word could be encoded by setting its associated entry equal to one and all other entries equal to zero. Though intuitive, this method is obviously inefficient for representing a language's entire vocabulary. Further, one-hot encodings do not capture the semantic or syntactic relationships between words, which are crucial for model generalization; rather, these encodings project words into a space where each word is equidistant and orthogonal to every other word in the vocabulary. An improvement would be an embedding method that projects language in a continuous space where distances between embeddings are representative of semantic and syntactic relationships. In a well known example of such a space, performing "King - Man + Woman" using each word's embedding should result in a vector very near to the embedding of "Queen" [73]. The following section in the literature review will be dedicated to exploring methods of generating such language embeddings.

2.3.1 Creating Embeddings

Researchers have long sought to create language embeddings using neural network architectures [74]. Bengio et al. use a feedforward neural network with a linear projection layer and a non-linear hidden layer to prove that language embeddings obtained from neural network architectures yield superior results when compared to other language modeling methods, like the once popular N-gram [75]. A downfall of this particular neural network architecture, and those built upon it, is that its complexity makes it impractical for training on large corpora. Work by Mikolov et al. allows for up-scaling by introducing two new architectures implemented in a model called word2vec: one that trains using CBOW and another that relies on continuous skip-gram [76]. The word "continuous" in each of these approaches indicates that they represent language in a continuous vector space as opposed to discrete N-gram spaces. Figure 2.2 displays both the CBOW and skip-gram architectures. Both approaches rely on local word co-occurances, meaning that they derive a word's representation based on its neighboring words. Using the CBOW approach, the neural network's training task is to predict a word given a range of words from its history (i.e., words to its left) and words from its future (i.e., words to its right). The example in Figure 2.2 shows CBOW using two history words and two future words. The ordering of the history and future words has no influence on the model's prediction, hence the name, "bag-of-words". The skip-gram approach operates in reverse; given a particular word, its training task is to predict words from a range of history and future words. Since the model only predicts one word at a time, a single input word will yield as many training examples as the number of selected history and future words. In Figure 2.2, for example, each of the four output history and future words would constitute a separate training example. The examples are sampled for training according to the distance between the input word and the output word; the greater the distance between the words, the less the example will be sampled for training.

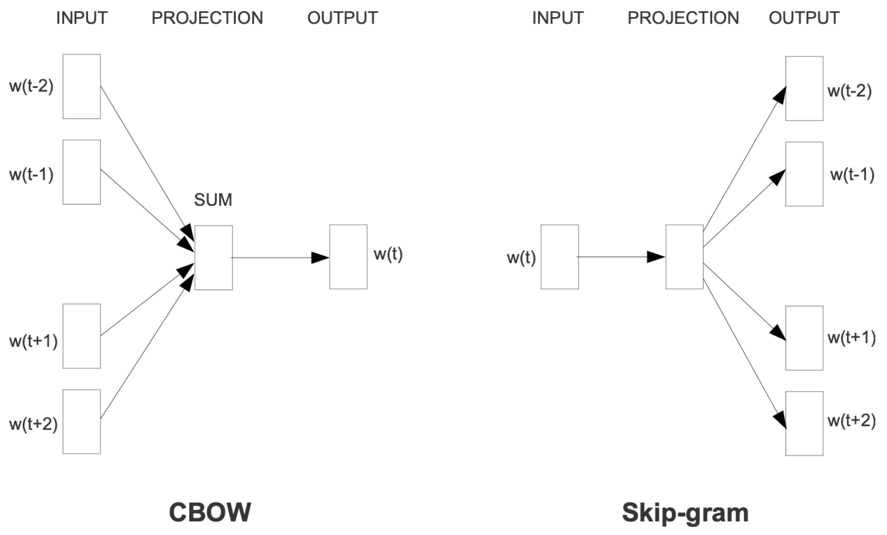


Figure 2.2: CBOW and Skip-gram Architectures for Word2Vec [76].

Either the CBOW or skip-gram approach can be used to build a word2vec model that can then be pre-trained on a large corpus to produce embeddings for each word in the corpus vocabulary. An alternative to word2vec's neural network approach that relies on local co-occurances is the global vectors for word representation model [77], which builds word representations using global word co-occurances with a weighted least squares objective. Once trained though, these models produce context-independent embeddings, meaning that the same word is always represented with the same vector, regardless of the context in which it appears. The Transformer architecture, introduced by Vaswani et al., resolves this issue through its self-attention mechanism, which creates context-dependent embeddings [78]. The contextual embeddings produced by self-attention deal with long-term dependencies better than the alternative LSTMs used in Embeddings from Language Models [79]. The creators of the Transformer intended to use it for machine translation applications, so it is composed of an encoder and decoder in the original setup. Researchers have since realized the power of each transformer component in other NLP applications, with the creation of the Generative Pre-trained Transformer [54] from transformer decoders, and BERT [80] from transformer encoders. This work relies on BERT to classify requirements and then extracts the resulting contextual requirement representations for use in downstream applications.

2.3.2 Bidirectional Encoder Representations from Transformers (BERT)

Upon its introduction, BERT demonstrated state-of-the-art results on various NLP tasks that include question answering, named-entity recognition, and, most important to this work, sequence classification [80]. The original BERT model comes in two sizes: $\text{BERT}_{\text{BASE}}$, with 12 transformer encoder layers and 12 attention heads for a total of 110M parameters, and the massive $\text{BERT}_{\text{LARGE}}$, with 24 transformer encoder layers and 16 attention heads for a total of 340M parameters. The encoder layers each contain a multi-headed attention mechanism and a feed-forward neural network. Each head of the attention mechanism projects a word embedding into a different contextual representation space before weighting and combining the representations into a new, balanced embedding [78]. The use of many layers allows BERT to develop a sophisticated understanding of language. A problem with models of this scale, however, is the shear amount of data and computational resources required for training. This obstacle is overcome through transfer learning, which is formally defined as follows: "given a source domain D_s and learning task T_S, a target domain D_T and learning task T_T, transfer learning aims to help improve the learning of the target predicting function $f_T(\cdot)$ in D_T using the knowledge in D_T and T_T, where $D_S \neq D_T$, or $T_S \neq T_T$" [81]. In other words, transfer learning involves training a model on a convenient task that improves the model's performance on a different task for which training is less convenient. In BERT's case this involves pre-training on unsupervised tasks to develop a general language understanding that can later be fine-tuned for a particular supervised task, which requires a labeled dataset. Specifically, BERT undergoes unsupervised pre-training on a "masked language model" task and "next sentence prediction" task applied to a corpus of 3,300M words. The masked language model task involves randomly replacing a word in an input sequence with a special token, designated "[MASK]" and then training the model to predict the replaced word. This task, adapted from the Cloze procedure [82], trains BERT to create word embeddings based on bidirectional context (i.e., based on both words to its left and its right), whereas previous transformer-based models were trained with a "next word prediction" task that could only consider a unidirectional context (i.e., either words to its right or to its left). The next sentence prediction task then involves inputting two sentences and having the model predict whether the two sentences occurred alongside one another in the source text. This task benefits BERT's performance when later fine-tuned to sentence-level tasks.

Figure 2.3 displays the BERT pipeline. To input a text sequence into the model, each word is first converted to a standard WordPiece embedding [83]. The embeddings are then run through BERT to create contextual embeddings that are used for the desired task. Most BERT implementations come pre-trained "out of the box" and can then be fine-tuned with a labeled dataset. The fine-tuning process adjusts all model parameters to match outputs for given inputs, and requires significantly less data than pre-training; one of the datasets used to fine-tune BERT in the original paper is smaller than the pre-training dataset by a factor of one million [80]. Publicly available labeled datasets, such as the Stanford Question Answering Dataset [84], exist for common tasks. Alternatively, users can fine-tune the model with a cus-

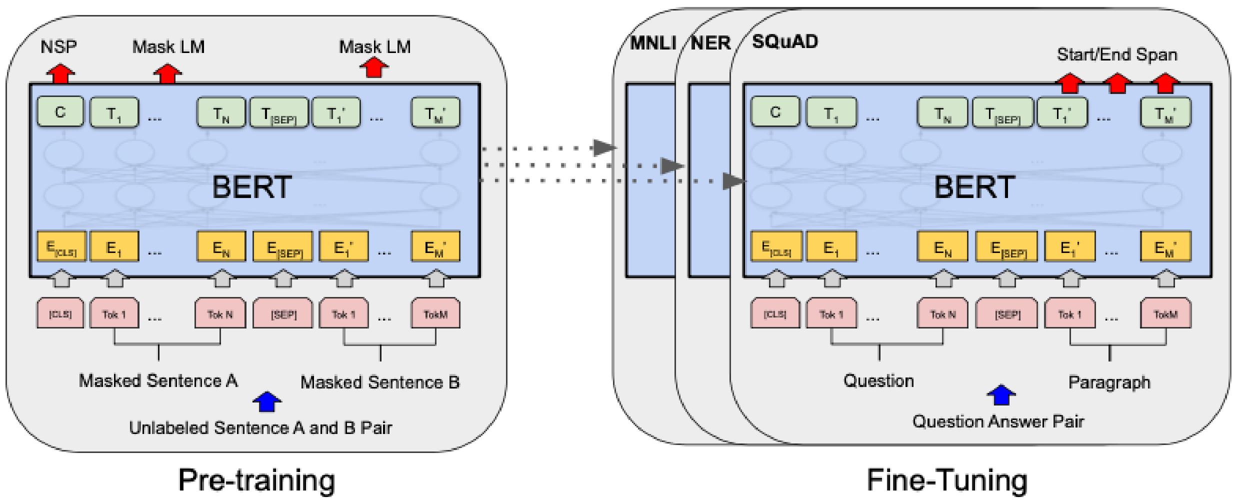


Figure 2.3: Pre-training and Fine-tuning BERT [80].

tom dataset. When fine-tuned for sequence classification, BERT uses a special token, designated "[CLS]" (which stands for "classification") and appended to the beginning of each input sequence, to represent the whole sequence for an output classification layer. The [CLS] token embedding only contains useful information after the model undergoes fine-tuning; otherwise, the [CLS] token embedding cannot be considered an adequate representation of the entire sequence. Alternatively, representations for entire sequences can be generated using Sentence Transformers[5], which trains BERT and other Transformer-based models specifically to create sequence embeddings.

2.4 Summary

Existing studies demonstrate that BERT, a state-of-the-art NLP model, can be a valuable tool for requirements management. In particular, BERT has shown promising results for requirement classification and evaluating requirement similarity. However, it remains unclear whether the BERT's requirement embeddings can distinguish between requirements from different engineering projects. It also remains unclear

[5]https://www.sbert.net/index.html

whether BERT can reliably distinguish between FRs and NFRs within engineering projects. This work seeks to provide clarity on both of these issues by fine-tuning BERT on two classification tasks: PDC, which classifies according to project, and FC, which classifies according to functionality. The performance of the fine-tuned models will indicate BERT's ability to differentiate requirements between and within diverse sets of design documents. Further, this work investigates how the fine-tuned model's [CLS] requirement embeddings perform in comparison to general-language SB sequence embeddings when applied to requirements management applications.

Chapter 3

Research Methods

3.1 Overview

This chapter explains each step in the research map presented in Figure 1.1. The chapter begins with a review of the requirements dataset used in this work. Then, Section 3.3 explains how requirements are labelled for the PDC task, where requirements are classified according to their originating project, and the FC task, where requirements are classified as either functional or nonfunctional. Section 3.4 presents the procedure for fine-tuning BERT on the requirement classification tasks as well as the Matthews correlation coefficient (MCC) used to evaluate model performance. Section 3.5 details the process of extracting and analyzing embeddings from the fine-tuned BERT models. Then, Section 3.6 explains the steps for identifying similar requirements using the PDC [CLS] embeddings, and Section 3.7 describes an application of the FC [CLS] embeddings for predicting requirement change propagation. Finally, Section 3.8 presents an alternative BERT model that will serve as a comparison to the models fine-tuned for requirement classification.

3.2 Review of the Requirement Corpus

Three of the dataset's five requirements documents come from private industry and have undergone previous analyses [11, 17, 23, 24, 31]. Project 1 involves creating a production line for threaded pipes, Project 2 focuses on the design of manufacturing equipment for exhaust gas flaps, and Project 3 entails the design of industrial textile equipment. Projects 4 and 5 are publicly available requirements documents for designing subsystems of the SKA[1], the world's largest radio telescope. Project 4 describes the design of a dish element, while Project 5 describes an artifact responsible for correlating and beamforming. Though this work does not claim this dataset to be a perfectly representative sample of all system requirements, the dataset seems diverse enough to provide, at the very least, an indication of BERT's ability to distinguish requirements in general.

Table 3.1: Dataset Statistics.

Project	Number of Requirements	Avg. Words Per Requirement	Vocabulary Size
1	350	19.7	1000
2	159	25.1	1379
3	214	27.6	1043
4	289	29.7	1342
5	291	40.6	1554
Total	1303	28.4	3765

[1]https://www.skatelescope.org/key-documents/

Table 3.1 displays statistics for each project's requirements document as well as statistics for the corpus as a whole. In total, the dataset contains 1,303 requirements with an average length of 28.4 words and a vocabulary size (i.e., the number of unique words) of 3,765. Project 2 has the fewest number of requirements, with 159, while Project 1 has the most requirements, with 350. Project 1 also contains the shortest requirements, whereas Project 5 contains the longest, on average. Similarly, Project 1 has the smallest vocabulary, and Project 5 has the largest.

3.3 Creating Labeled Datasets

Fine-tuning BERT for sequence classification requires a labeled dataset. A "sequence" is whatever string of text must be labeled (e.g., a movie review or a tweet); here, each requirement is considered a sequence and must be labeled for both classification tasks. The labeled datasets follow a tabular structure where each row is assigned to a requirement. A "Requirements" column contains the requirement text and a "Label" column contains the corresponding label. As will become evident, the effort required to provide labels differs substantially between the two requirement classification tasks.

3.3.1 PDC Task

To fine-tune BERT for PDC, each requirement must be labeled accordingly. All requirements from Project 1, for example, are provided a label of "Doc1," and so on. Providing these labels is trivial; if each document has a corresponding Excel spreadsheet or Pandas DataFrame (tabular data structure in Python) where each row contains a requirement, then labeling simply involves appending a new column with the document name in each cell. While the ease of labeling is an advantage of this classification task, this labeling scheme strictly prevents the fine-tuned model from generalizing to requirements documents outside of the training dataset. The model here is only trained on five documents, meaning it can only output labels for those five documents. If presented with a requirement from a sixth document, for instance, the fine-tuned model would incorrectly provide a label for one of the five training documents.

3.3.2 FC Task

To create the dataset for FC fine-tuning, each requirement is manually labeled as either "functional" or "nonfunctional". As covered in Section 2.2.2, the distinction between these classes is not always clear, so one individual's labels may differ from another's. Therefore, some preliminary analysis is required to verify the assigned labels. As a reminder, the following definitions are used in this work to differentiate between FRs and NFRs:

- **FR**: "what a product must do, be able to perform, or should do [30]"
- **NFR**: all requirements other than FRs

Though the NFR definition is admittedly weak when compared to alternative definitions, it is used here to create a binary classification task that encompasses all requirements. The primary researcher provided labels for all 1,303 requirements in the dataset. A breakdown of the labels is presented in Table 3.2. In total, around 70% of the dataset requirements are labeled as nonfunctional. Document 5 has the most balanced distribution, with a near 50/50 split between functional and nonfunctional, while Document 2 has the least balanced, containing a single FR.

Table 3.2: Label Counts for FC Task.

Project	FRs	NFRs
1	123	227
2	1	158
3	42	172
4	81	208
5	144	147
Total	391	912

To verify these labels, a stratified, randomly generated, 10% sample of the dataset was labeled independently by two other individuals, both of whom work in the same lab as the primary researcher. These individuals were sent an email that explains the labeling task and provides the requirement sample. A copy of the instructions can be found in Appendix F. Only a sample is used for comparison due to the extensive labor required to label the entire dataset. Stratified random sampling ensures that the sample maintains the same document proportions as the overall dataset. Cohen's kappa statistic [85] is then used to evaluate inter-rater reliability between the primary researcher and each individual. Unlike percent agreement, the kappa statistic, κ, takes into account the possibility that raters may agree by chance. Equation 3.1 displays the Cohen's kappa statistic formula, where $P(a)$ is the actual agreement among raters, and $P(e)$ is the expected agreement due to chance.

$$\kappa = \frac{P(a) - P(e)}{1 - P(e)} \tag{3.1}$$

Here, the kappa statistic is computed using the Scikit-learn package [86]. Evaluating the first and second individual's labels against the primary researcher's labels results in Cohen's kappa statistics of 0.73 and 0.82, respectively.

Table 3.3: Interpretation of Cohen's Kappa Statistic [87].

Value of Kappa	Level of Agreement
0 – 0.19	None
0.20 – 0.39	Minimal
0.40 – 0.59	Weak
0.60 – 0.79	Moderate
0.80 – 0.89	Strong
0.90 – 1.00	Almost Perfect

Table 3.3 presents McHugh's interpretations of the kappa statistic [87]; these interpretations do not exactly align with those proposed by Cohen, but are used here since they are more conservative than Cohen's. Following McHugh's interpretations, the computed statistics show a moderate and a strong level of agreement. Out of the 130 total requirements in the sample, the first individual's label did not match the primary researcher's label for 15 requirements, and the second individual's label did not match for 10 requirements. Taking a closer look at the requirements where the labels did not match, some contain both functional and nonfunctional aspects. Consider the following requirement from Document 4:

> *"It shall be possible to locate two Dishes with centre-to-centre spacing of 30 meters without any possibility of collision."*

Though this requirement seems to describe a function for locating two dishes, it could instead be interpreted as focusing on the specifications or reliability of the function, rather than on the function itself. Regardless of ambiguities, the kappa statistics indicate general agreement on the distinction between FRs and NFRs, suggesting that the primary researcher's labels follow a coherent pattern. This work's results will indicate whether fine-tuned BERT can replicate the labeling pattern. The model fine-tuned on the FC task should be able to generalize to requirements documents outside of the training set, since any given definition of functional and nonfunctional is universal across requirements documents. The exact definitions of functional and nonfunctional may vary, but the results from this dataset will indicate BERT's general ability to distinguish requirement functionality.

3.4 Fine-Tuning for Requirement Classification

Once labeled, the datasets are shuffled and split as follows: 80% of the requirements are allocated as a training set and the remaining 20% are reserved as a test set. The training set is further broken down into 90% for training and 10% for validation. Similar to BERT's pre-training corpus, the requirement text does not undergo pre-processing (e.g., removal of stopwords) other than tokenization, where words are split into the "subtokens" that comprise BERT's pre-trained vocabulary. Subtokens are typically pieces of words and are the basic units of language that BERT represents with embeddings. Additionally, the class labels are converted into integers ranging from zero to $[number\_of\_classes] - 1$ (e.g., a parent document label of "Doc 1" becomes "0"). This work uses the "bert-base-uncased" ("uncased" signifies that letter casing is ignored) model in the Hugging Face [88] implementation of BERT for Sequence Classification. This particular model consists of 12 encoder layers, 12 self-attention heads, and an embedding size of 768. Fine-tuning the model for each requirement classification task occurs over three epochs of the 937 test set examples with a batch size of 16, AdamW optimizer with a learning rate of 2e-5, warmup ratio of 0.1, and weight decay of 0.01. During training, the model updates parameters to minimize cross-entropy loss, which maximizes the probability of correct classification. This procedure has a total runtime of 124 seconds for the PDC task and 97 seconds for the FC task when executed in a Google Colaboratory session equipped with a 16GB NVIDIA P100 GPU. While the stated hyperparameters fall into the ranges recommended by BERT's creators, optimal values will vary depending on the dataset and may require adjustment when fine-tuning on a different set of requirements documents.

3.4.1 Evaluating Classification Performance

This work relies on the MCC metric to evaluate the performance of the fine-tuned models. The MCC evaluates model performance across all classes, even with varying class sizes [89]. Given that both requirement classification tasks have an unbalanced dataset (i.e., varying number of samples in each class), the MCC is more informative than simple performance metrics, like accuracy. An MCC of one indicates perfect prediction, while an MCC of zero indicates prediction equivalent to random guessing. Equation 3.2 displays the MCC formula for the binary classification case, where x contains the true class labels, y contains the predicted class labels, $Cov(x, y)$ is their covariance, and t_p, t_n, f_p, and f_n are, respectively, the number of true positives, true negatives, false positives, and false negatives within y.

$$\text{MCC}_\text{b} = \frac{Cov(x,y)}{\sigma_x \cdot \sigma_y} = \frac{t_p \cdot t_n - f_p \cdot f_n}{\sqrt{(t_p + f_p)(t_p + f_n)(t_n + f_p)(t_n + f_n)}} \tag{3.2}$$

Equation 3.2 indicates that the MCC is equivalent to measuring the correlation between x and y [90]; in fact, MCC is simply a special case of the Pearson correlation coefficient. While Equation 3.2 suffices for the FC task, the PDC task has five classes and requires the general multiclass calculation shown in Equation 3.3, where K is the number of classes, s is the number of total samples, c is the number of correctly predicted samples, p_k is the number of times class k was predicted, and t_k is the number of times class k truly occurred.

$$\mathrm{MCC_m} = \frac{s \cdot c - \sum\limits_{k=1}^{K} p_k \cdot t_k}{\sqrt{(s^2 - \sum\limits_{k=1}^{K} p_k^2)(s^2 - \sum\limits_{k=1}^{K} t_k^2)}} \tag{3.3}$$

Here, MCC is computed by setting it as the default evaluation metric in the Hugging Face implementation of BERT. The model then automatically returns MCC whenever it processes a test set. It should be noted that while the MCC provides a useful, single-number evaluation of model performance, it does not capture all of the information contained in a confusion matrix, which contains counts of correct and incorrect predictions for each class. Therefore, confusion matrices are provided in addition to the MCC for each requirement classification task. MCC provides an easily interpretable evaluation of performance, while the confusion matrix provides a more detailed, but less concise, performance evaluation.

3.5 Obtaining and Analyzing Requirement Embeddings

After fine-tuning on a requirement classification task, BERT has learned requirement-specific embeddings that may prove useful in requirements management applications. This section explains the applied methodology for extracting and analyzing embeddings from the fine-tuned models. The embedding analysis consists of a visualization of the embedding space followed by a numeric evaluation of requirement relationships.

3.5.1 Extracting Embeddings from Fine-Tuned Models

As covered in Section 2.3.2, BERT's [CLS] token embedding represents an entire input sequence prior to classification. Logically, BERT attempts to learn the [CLS] embedding that yields the highest performance on the given fine-tuning task. Provided the fine-tuned model yields acceptable classification performance, the [CLS] embedding must contain information useful for making judgments about the input sequence. In the model resulting from the parent PDC task, the [CLS] embedding could be an advantageous representation for distinguishing requirements between projects, while the [CLS] token from the FC model could similarly be an effective representation of requirement functionality.

Extracting the [CLS] embeddings from the fine-tuned models is relatively straightforward using the Hugging Face implementation of BERT. This implementation has an option to return hidden states along with the model's predicted labels. "Hidden states" refer to the initial WordPiece embeddings as well as the outputs from each of BERT's transformer encoder layers. The Hugging Face documentation indicates the size of each layer's hidden states to be [batch_size, sequence_length, hidden_size]. The last layer contains the final [CLS] embedding used for classification. To obtain this embedding, each requirement in the dataset is individually input into the fine-tuned model, which then outputs the hidden states corresponding with the requirement. The last layer (13th for $\text{BERT}_{\text{BASE}}$ since initial WordPiece embeddings are included) of the hidden states is selected, then the first item in the batch (the batch_size here is one since requirements are input individually), then the first item of the sequence (the [CLS] token is always appended to the beginning of each sequence). Following this procedure for each requirement yields a set of [CLS] embeddings for the entire dataset.

3.5.2 Analyzing Requirement Relationships

The [CLS] embeddings extracted from the fine-tuned models are not easily interpretable as vectors with length of 768. To provide a general idea of the embedding spaces, dimensionality reduction techniques are first applied to visualize the embeddings. Then, requirement relationships are quantified by comparing embeddings with the cosine distance metric.

Visualizing Requirement Relationships

T-distributed stochastic neighbor embeddings (t-SNE) are used here to project the [CLS] vector, of dimension 768, into two-dimensional space for visualization. It is important to note that t-SNE plots do not preserve Euclidean distance between data; rather, the t-SNE algorithm minimizes a cost function, displayed in Equation 3.4, equal to the Kullback-Leibler divergence, KL, of the joint probabilities of the original data, P, and the two-dimensional projections, Q. Here, p_{ij} is the joint probability between points x_i and x_j according to a Gaussian distribution representing the high-dimensional space and q_{ij} is the joint probability between points y_i and y_j according to a Student-t distribution representing the two-dimensional space. The procedure for computing p_{ij} and q_{ij} is not presented here but can be found in Appendix A of the original t-SNE paper by van der Maaten [91].

$$C = KL(P||Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}} \tag{3.4}$$

Minimizing this cost function using gradient descent algorithms results in two-dimensional embeddings that reflect general trends in the higher-dimensional data. However, the cost function is not convex, so results may vary slightly for different initializations. This work relies on the Scikit-Learn [86] implementation of t-SNE, which automatically applies principal component analysis, a dimensionality reduction technique, as a preprocessing step to improve the quality of the resulting embeddings. Hyperparameters, such as perplexity and learning rate, are kept at their default values.

Quantifying Requirement Relationships

Cosine similarity is commonly used to represent two vectors' relationship, and is applied here to determine a requirement's nearest neighbors in the [CLS] embedding space. Results from a nearest neighbors search are easier to interpret with positive-valued distances, so cosine similarity is converted to cosine distance. The conversion involves simply subtracting cosine similarity from one. Whereas cosine similarity is bounded by $[-1, 1]$, cosine distance is always a positive number bounded by $[0, 2]$. Equation 3.5 presents the formula for cosine distance, where **A** and **B** are two vectors with angle, θ, between them.

$$\text{cosine distance} = 1 - cos(\theta) = 1 - \frac{\mathbf{A} \cdot \mathbf{B}}{||\mathbf{A}|| \, ||\mathbf{B}||} \tag{3.5}$$

It should be noted that cosine similarity, and by extension cosine distance, are not "true" metrics since they do not satisfy the triangle inequality. However, cosine distance is preferred to alternatives like Euclidean distance for several reasons. First, cosine similarity is more commonly used within NLP. For example, the SB model uses a cosine similarity loss function when fine-tuned to compute semantic textual similarity [92]. In the embedding space, relative angles generally seem to capture text similarity better than magnitudes. Further, it has been shown that Euclidean distance becomes less meaningful in higher-dimensional spaces [93]. Lastly, cosine similarity's fixed boundaries make it more readily interpretable than unbounded distance metrics that could become inflated by arbitrary scaling.

3.6 Identifying Similar Requirements

Since the BERT model fine-tuned for PDC has developed [CLS] embeddings specifically designed to distinguish between requirements from different documents, these embeddings could provide a measure of the similarity between project requirements. The rationale here is that BERT will place similar requirements in similar areas in the [CLS] embedding space. There are no ground-truth similarity measures for this dataset, so the findings will be evaluated against intuition as well as results computed with a semantic textual similarity SB model. Similarity is observed at two levels: the document level and the individual requirement level. The document-level analysis could aid designers in evaluating the overlap between projects as a whole, while a requirement-level similarity search could identify specific opportunities for the reuse of design solutions.

3.6.1 Similarity of Requirements Documents

To begin exploring similarity at the document level, all embeddings are visualized in a t-SNE plot that is color-coded to reflect each requirement's parent document. This visualization should provide some indication as to how well the [CLS] embeddings can distinguish between parent documents and which parent documents are most similar to one another. The similarity amongst documents is further explored by measuring the cosine distance between the average [CLS] embedding of each document's requirements. Averaging the [CLS] embeddings for all requirements in a document produces a single, document-level embedding that is presumed to represent the project as a whole.

3.6.2 Requirement Similarity Search

To find similar requirements individually, the [CLS] embedding for a requirement of interest is input to the Facebook Artificial Intelligence Similarity Search (Faiss) [94] algorithm, which then searches among the set of [CLS] embeddings of all other requirements. Requirements with the same parent document as the requirement of interest are omitted from the search since the objective is to identify similar requirements in other projects that could be reused. Here, the cosine distance metric is used to find the three nearest neighbors to the query requirement. The Faiss algorithm in particular is used because of its ability to efficiently search for nearest neighbors among dense, high-dimensional vectors. Though it does not directly support searches based on cosine similarity, Faiss does support inner product searches. By first normalizing the [CLS] embeddings by their magnitude (i.e., placing them all on the unit sphere), the inner product search becomes equivalent to a cosine similarity search. Results are then presented in terms of cosine distance.

3.7 Predicting Requirement Change Propagation

As reviewed in Section 2.2.4, requirement change can be predicted by requirements' semantic similarity. It has also been established that change propagates differently among FRs than it does among NFRs. The

[CLS] embeddings obtained from BERT fine-tuned for FC not only represent requirements' semantic meaning, but also their functionality. Therefore, these [CLS] embeddings could be used in a process that builds upon previous research by predicting change according to functional and nonfunctional relationships. Documented ECNs from Projects 2 and 3 are used to explore whether the FC [CLS] embeddings can predict requirement change. Each ECN lists the requirements that it affects; requirements effected by an initial ECN are used to predict which requirements will be affected by subsequent ECNs. Here, the FC [CLS] embedding of the initially affected requirement is used as a query, and all remaining requirements in the document are sorted according to their cosine distance from the query requirement. The Faiss algorithm is used to perform this sorting by setting the search's number of nearest neighbors equal to the number of requirements in the document. A requirement's location in the sorted list is referred to as its ranking. If this procure is capable of predicting requirement change, then requirements affected by future ECNs should be highly ranked. For comparison, the results are assessed against those generated with SB embeddings given the same ECNs.

3.8 Obtaining SB Embeddings

The [CLS] embeddings generated from BERT fine-tuned for requirement classification have been formed specifically to represent textual requirements for the assigned classification task. The application of the [CLS] embeddings to downstream tasks, such as identifying similar requirements and predicting change propagation, is inspired by the transfer learning paradigm; in the same way that BERT's pre-training tasks give it a general language understanding useful for downstream tasks, it is hypothesized that the requirement classification tasks give BERT a specific understanding of requirements that could improve its performance in later requirements management applications. To test this hypothesis, the performance of the [CLS] embeddings generated by the fine-tuned models is compared to that of an SB model, which is a type of Sentence Transformer model. Though it may seem logical, the fine-tuned [CLS] embeddings cannot simply be compared to their values prior to fine-tuning since they do not meaningfully represent the input sequence at that stage. The Sentence Transformer models are based on work that trains BERT to create embeddings specifically for sequences of text, rather than individual tokens [92]. The particular model used here is "multi-qa-distilbert-cos-v1," which is built on DistilBERT (a smaller, *distilled* version of BERT) and fine-tuned on a dataset of 215M question-answer pairs to identify, via cosine similarity, text relevant to a given query. Since both requirements management applications rely on textual similarity, this SB model should serve as a good baseline for judging the performance of the [CLS] embeddings.

Chapter 4

Results

4.1 Overview

Section 4.2 presents the results of BERT fine-tuned on the PDC task. Then, the resulting [CLS] embeddings are applied to identify similar requirements at the document level in Section 4.2.2 and at the individual requirement level in 4.2.3. Both of these sections also include a comparison to results generated by the SB model. Section 4.3 presents the results of BERT fine-tuned on the FC task, and Sections 4.3.2 and 4.3.3 apply the resulting [CLS] embeddings to predict requirement changes documented in ECNs from Projects 2 and 3. The change prediction results are also compared to those generated by the SB model.

4.2 BERT Fine-Tuned for PDC

Table 4.1 displays training and validation metrics for each epoch of training. The training loss is computed on the training set, while the validation loss and MCC are computed on the validation set, which is held out from training and used to track model performance on data outside the training set. As a reminder, training and validation loss are computed via a cross-entropy loss function, which is standard for classification tasks. Table 4.1 indicates that both the training and validation loss decrease with each additional epoch, while the MCC increases, indicating model improvement on the PDC task.

Table 4.1: PDC Training Metrics.

Epoch	Training Loss	Validation Loss	MCC
1	1.307	0.835	0.719
2	0.610	0.342	0.952
3	0.323	0.232	0.976

Table 4.2 presents a confusion matrix that indicates the fine-tuned model's performance on the test set. The model's predicted parent document for each requirement is indicated along the matrix columns, while the actual parent documents are indicated along the rows. With perfect performance, the predicted parent document would always match the actual parent document, and all off-diagonal values in the matrix would be equal to zero. The correctly classified requirements are indicated along the matrix diagonal in green, and the misclassifications, at off-diagonals, are indicated in red. With the exception of the four Document 2 requirements that were misclassified as Document 4 requirements, the confusion of one parent document for another only occurs in quantities of one or two requirements. The model's classification performance on a particular document can be understood through precision and recall. Precision indicates, as a percentage, how often the model is correct when it predicts a specific class label. Precision can be calculated down a column by dividing the number highlighted in green by that column's total. Recall indicates, also as a percentage, how often a model can correctly predict the class label when given examples from a particular class. Recall can be calculated along a row by dividing the number highlighted in green by that row's total. The model has its lowest precision, 91%, for Document 4, and its lowest recall, 84%, for Document 2. Both Documents 2 and 5 have perfect precision, while Document 4 has perfect recall. The fine-tuned model's overall performance is evaluated with MCC, which is calculated to be 0.95. This score is associated with very high correlation between predicted labels and actual labels [95] and sug-

gests that the fine-tuned model can reliably distinguish requirements from different documents.

Table 4.2: Confusion Matrix for PDC BERT Model on Test Set.

Actual Parent Document \ Predicted Parent Document	Doc1	Doc2	Doc 3	Doc4	Doc5	**Totals**
Doc1	69	0	0	1	0	70
Doc2	1	36	2	4	0	43
Doc3	2	0	30	0	0	32
Doc4	0	0	0	58	0	58
Doc5	0	0	0	1	57	58
Totals	72	36	32	64	57	261

4.2.1 Exploration of the PDC Embedding Space

Equipped with a model fine-tuned on the PDC task, [CLS] embeddings of all requirements in the dataset can now be retrieved from the model's hidden states. The [CLS] embeddings are visualized as a t-SNE plot in Figure 4.1, where requirements are colored according to their true parent document label. It is important to note that since the entire dataset is included, misclassifications among the training and validation set are now evident; these misclassifications are absent from the confusion matrix in Table 4.2, which only includes results for the test set. Despite misclassifications, the model manages to form a cluster for each document. The Document 1 cluster contains requirements from Documents 2 and 3, and the Document 2 cluster contains requirements from Documents 1 and 3. Overall, Documents 1 and 5 appear to be the most distinct from one another, while Documents 3 and parts of Documents 2 and 4 have some overlap. The overlap might consist of requirements that address a similar topic, such as safety regulations, within each document.

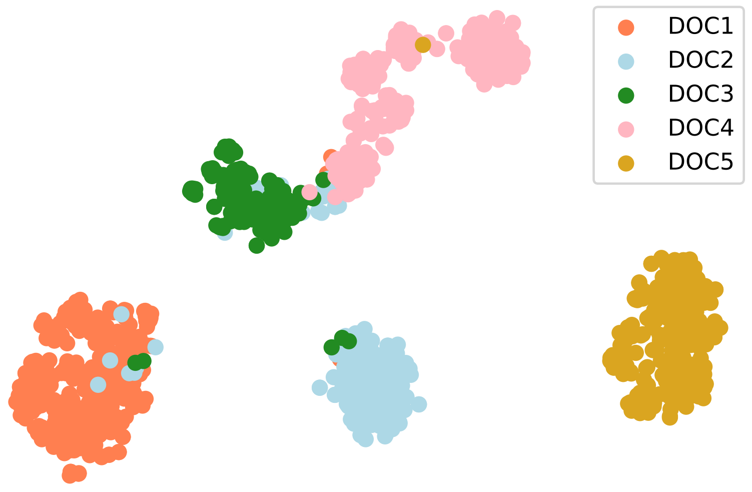
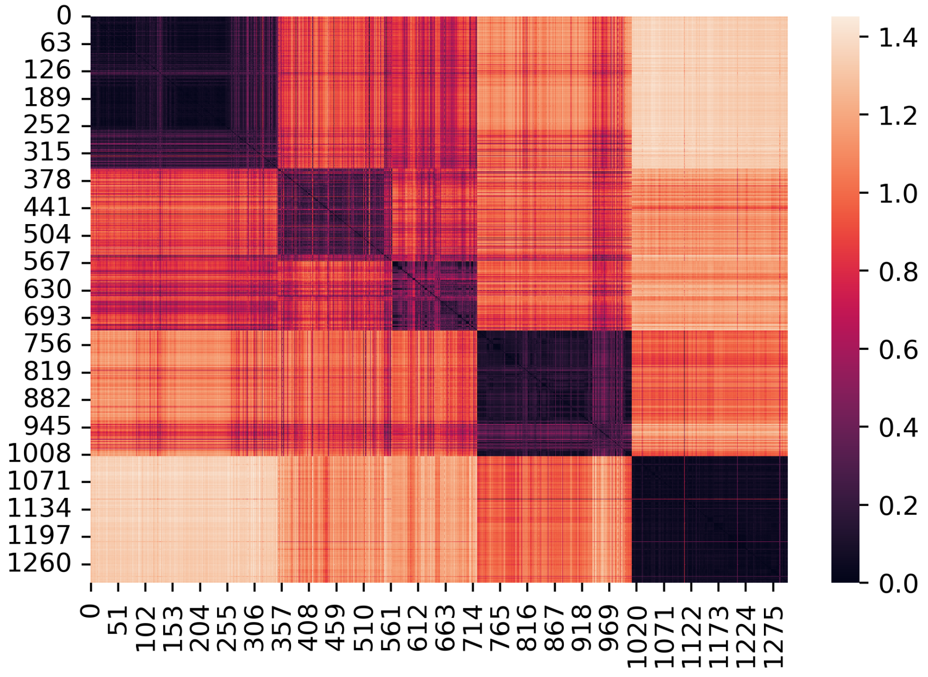


Figure 4.1: t-SNE Plot of [CLS] Embeddings Obtained from PDC Task.

To investigate the embedding space further, heatmaps, shown in Figure 4.2, of cosine distance matrices for [CLS] embeddings are generated. Requirements are generally organized by topic within requirements documents, so the heatmaps could indicate sections of topical overlap between documents. In Figure 4.2a each row and column of the matrix corresponds with a requirement, which are in order beginning with Document 1's first requirement and ending with Document 5's last requirement. The cell colors within the symmetric matrices represent the cosine distance between each pair of requirements. The boundaries between each document are roughly identifiable by the five large, dark squares along the matrix diagonal. Documents 1, 2, and 3 appear to form a set of requirements fairly distinct from Document 4 and further yet from Document 5. Lightly colored squares, visible in the lower left and upper right corners of the matrix, indicate Document 1 and Document 5 to have the greatest cosine distance between one another. An interesting feature of the colormap is the dark band visible towards the end of Document 4, roughly between rows 945 and 1008.


(a) Heatmap for Complete Dataset.

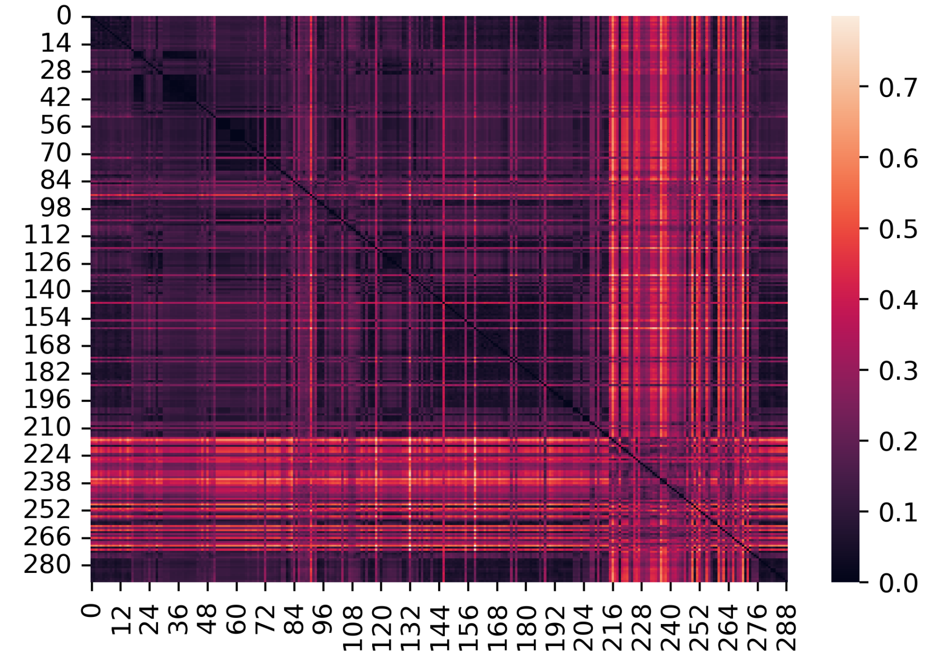

(b) Heatmap for Document 4.

Figure 4.2: Heatmaps of Cosine Distances Between PDC [CLS] Embeddings.

The requirements in this band appear to differ from the rest of Document 4's requirements; out of all the requirements in Documents 4 and 5, this band contains the requirements that are closest to those from Documents 1, 2, and 3. For a more detailed look at this band, Figure 4.2b displays a cosine distance matrix of Document 4 requirements only. Note that Figure 4.2b's color scale is recalibrated from that shown in Figure 4.2a to emphasize details. Here, the band is shown to contain around 55 requirements, ranging approximately from row 215 to row 270. According to the section headings in the original PDF version of Document 4, the majority of requirements in this range pertain to "Repair and Replacement," "Manufacturing Data Packs," "Packaging, Handling & Transportation," and "Safety and Security." These topics contrast those seen in the rest of Document 4, which is mainly composed of technical specifications related to the function and operation of the SKA's dish element. Conversely, these topics seem to overlap with those frequently found in Documents 1, 2, and 3, which all detail the production of manufacturing equipment. Overall, this exploration of the parent document embedding space has shown that while each document's requirements are represented distinctly, their distance from one another can indicate topical overlaps.

Comparison with SB Embedding Space

The SB embedding space is explored here to make evident the differences between the parent document embedding space and that of a model trained to create embeddings for general text sequences. Figure 4.3 displays a t-SNE plot of the complete dataset using embeddings generated by the SB model. Documents appear clustered, though not as distinctly as they are in Figure 4.1. Documents 1, 2, and 3 form a larger cluster, which includes some requirements from Document 4 as well. Once again, Document 5 emerges as the most distinct document.

A heatmap of the cosine distance matrix is shown in Figure 4.4. In contrast to the range of distances shown in Figure 4.2a, the SB embeddings place requirements almost equidistant to one another. There are exceptions, however, in groups of requirements located close to one another in Documents 1 and 5. Upon close inspection, faint lines can be seen that approximately mark the transition from Document

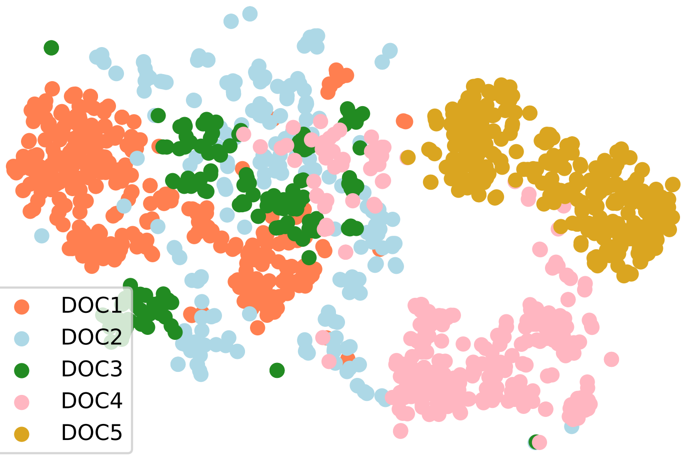


Figure 4.3: t-SNE Plot of Documents in SB Embedding Space.

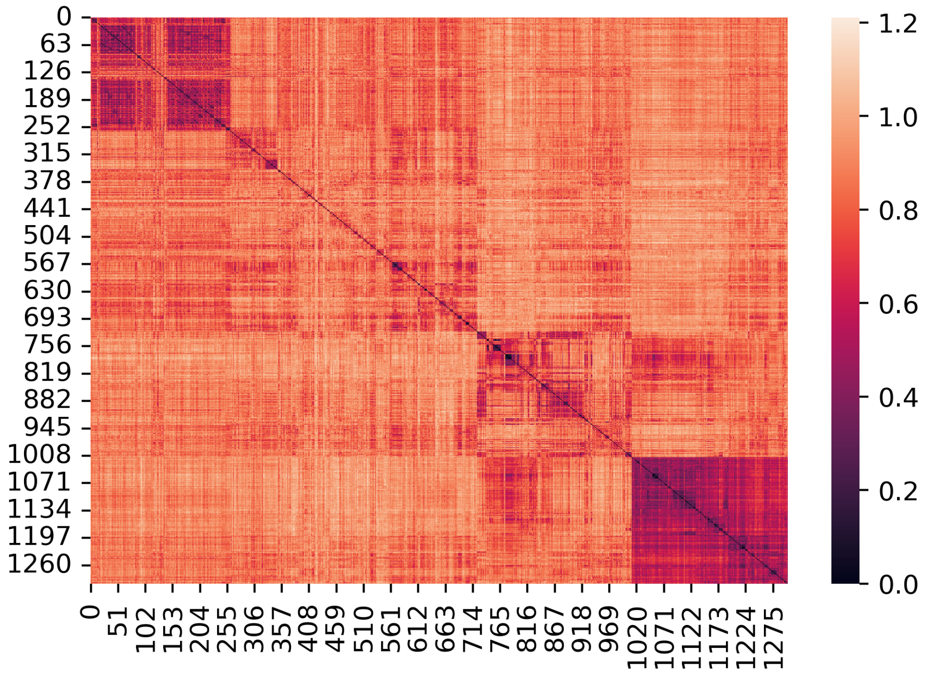


Figure 4.4: Heatmap of Cosine Distances Between SB Embeddings.

3 to Document 4 as well as the transition from Document 4 to Document 5. In all, the SB embeddings do appear to capture patterns that differentiate requirements between documents, but to a lesser degree than the [CLS] embeddings extracted from the PDC model.

4.2.2 Document Level Similarity

Based on the presented exploration, it is clear that the parent document model's [CLS] embedding space differentiates between requirements documents. This section applies the [CLS] embeddings to quantify the cosine distance between documents. Representations for documents as a whole are obtained by averaging the [CLS] embeddings for all requirements in a document. A cosine distance matrix is shown in Table 4.3 for the averaged embeddings of all five documents. Documents 2 and 3 are separated by the smallest cosine distance of 0.62, while Document 1 is the next nearest to Documents 2 and 3 with cosine distances of 0.79 and 0.65, respectively. Documents 3 and 4 have the closest remaining relationship with a cosine distance of 0.86. Document 5 is the most distinct, with cosine distances of at least 1.0 separating it from all other documents. Documents 1 and 5 are separated by the most significant observed distance of 1.39.

Table 4.3: Cosine Distance Between Averaged PDC [CLS] Embeddings.

	Doc1	Doc2	Doc 3	Doc4	Doc5
Doc1	0	0.79	0.65	1.02	1.39
Doc2	0.79	0	0.62	0.88	1.16
Doc3	0.65	0.62	0	0.86	1.22
Doc4	1.02	0.88	0.86	0	1.00
Doc5	1.39	1.16	1.22	1.00	0

As previously stated, there is no ground-truth to evaluate the results against, but there does appear to be sound backing for a couple of high-level observations. First, Projects 1, 2, and 3 are closer to one another than they are with Projects 4 and 5. This relative proximity is expected since Projects 1, 2, and 3 all describe the design of manufacturing equipment. Next, even though Document 5 is widely separated from all documents, it is closer to Document 4 than it is to Documents 1, 2, and 3, which makes sense given that Documents 4 and 5 originate from the SKA project. It is surprising, however, that despite coming from the same project, Documents 4 and 5 are separated by a cosine distance of 1.00. A potential explanation for such a large distance is that individual subsystems may have little relation or interaction in the design of a system as immense and complex as the SKA. In fact, Document 4 makes no mention of the correlator and beamformer described in Document 5. Searching Document 5 for the dish element described in Document 4 reveals that it was mentioned in only 4 of the 289 requirements. The band of requirements discussed in Section 4.2.1 may be partly responsible for Document 4 being further from Document 5 than expected and closer to Documents 2 and 3. A possible cause for Document 5 being so distinct from other documents, in general, is that it explicitly states its product's name, "CSP_Mid.CBF," in every requirement, while other documents only mention product names in some of their requirements. In all, this procedure for computing document similarity appears to reflect logical relationships between requirements documents while also unveiling similarities that may not be immediately obvious to a designer reviewing documentation.

Comparison to SB Results

The same document-level similarity analysis is performed with the SB embeddings for comparison. The cosine distance matrix for the documents' averaged embeddings is displayed in Table 4.4. The cosine distances computed with the SB embeddings are generally lower than those computed with the PDC model's [CLS] embeddings. The smallest cosine distance, with a value of 0.26, is observed between Documents 1 and 2 as well as Documents 2 and 3. Documents 1 and 3 are the next closest, with a cosine distance of 0.26. Document 5 is once again the most distinct and is separated from Documents 1, 2, and 3 by roughly the same amount, with cosine distances of 0.72, 0.73, and 0.70, respectively.

Table 4.4: Cosine Distance Between Averaged SB Embeddings.

	Doc1	Doc2	Doc 3	Doc4	Doc5
Doc1	0	0.26	0.32	0.58	0.72
Doc2	0.26	0	0.26	0.42	0.73
Doc3	0.32	0.26	0	0.57	0.70
Doc4	0.58	0.42	0.57	0	0.52
Doc5	0.72	0.73	0.70	0.52	0

The general trends observed in Table 4.4 remain mostly consistent with those observed in Table 4.3. Documents 1, 2, and 3 are once again closer to each other than to Documents 4 and 5. Additionally, Document 4 is the closest of all documents to Document 5. Some departures from the previously observed patterns are evident, such as Document 2, rather than Document 3, being closest to Document 4. While the SB embeddings do appear to capture many of the same document relationships as the PDC [CLS] embeddings, they create less distinction between documents. Without designating each document a particular area in the embedding space, it is unclear whether a computed distance is reflective of document relationships or simply due to chance.

4.2.3 Requirement Level Similarity

In this section, the PDC [CLS] embeddings are applied to identify similar requirements individually. Figure 4.5 demonstrates the search pipeline, which relies on the FAISS algorithm, with an example search from the dataset. For this example, the input requirement was selected from Project 3 since it is the most centralized, with an average cosine distance of 0.84 to the remaining documents. Searching from a centralized project should yield relevant results. The input requirement in the example specifies a safety feature for ease of lubrication. The search algorithm returns three requirements from Document 2. With a cosine distance of 0.36, the nearest requirement declares that wiring must satisfy the relevant safety regulations. The following result, with a cosine distance of 0.38, states that perishable tooling should be easy to remove and install. Also with a cosine distance of 0.38, the final result states that equipment should have an ergonomic design. Ergonomics is typically considered a subcategory of safety, with its purpose being to reduce injuries that may develop over long periods of work. As indicated by the highlighted text in Figure 4.5, the first and third results relate to the safety aspect of the input requirement, while the second result relates to the input through the concepts of removal and ease of maintenance.

This example demonstrates this procedure's ability to search multiple documents and find related requirements. The relevance of some results is not always initially obvious, but further inspection can reveal relationships that designers may find useful. However, requirements are sometimes unique to their project, and searches may fail to produce relevant results at all. The relevance of results depends on the span of projects that comprise the dataset. If the projects are closely related, searches find more relevant requirements than they could with vastly different projects. The relationship between projects plays an especially prominent role since the PDC BERT model embeds requirements with the objective of distinguishing parent documents. There could be similar requirements in other documents that are not considered simply because those documents are further from the input requirement's parent document. For example, all three results in Figure 4.5 come from Project 2. While it is possible that Project 2 actually contains the three requirements most similar to the input requirement, it could also be the case that only Project 2 requirements are returned due to Document 2's proximity to Document 3.

Comparison to SB Results

Using the same input requirement, a similarity search is conducted via the SB embeddings instead of the PDC [CLS] embeddings; the search results are displayed in Figure 4.6. The first two results belong to Document 4 and are separated from the input by cosine distances of 0.33 and 0.40, respectively. Both relate explicitly to safety, with the first indicating the need to mark hazardous equipment appropriately

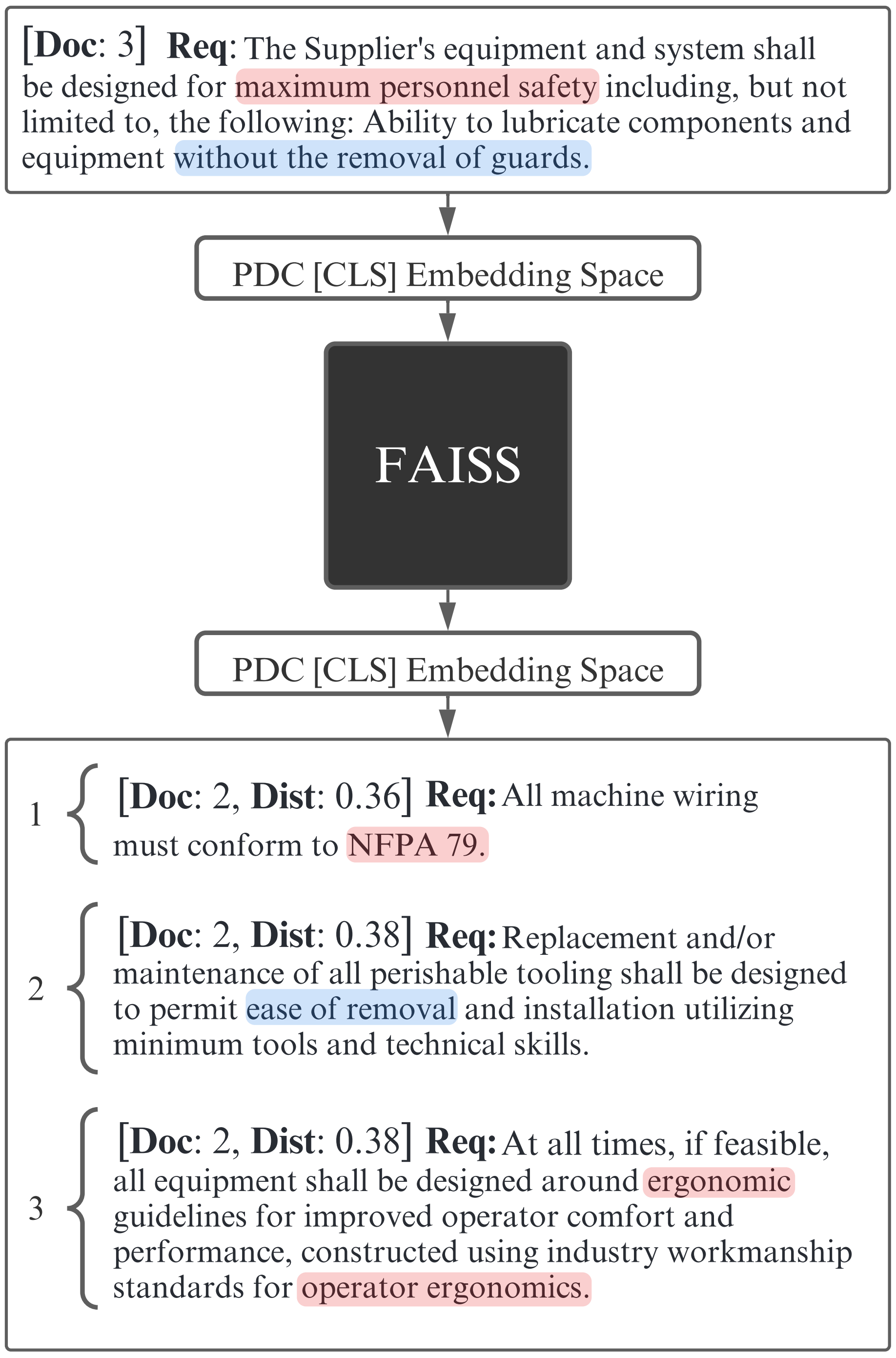

[Doc: 3] Req: The Supplier's equipment and system shall be designed for maximum personnel safety including, but not limited to, the following: Ability to lubricate components and equipment without the removal of guards.
PDC [CLS] Embedding Space
FAISS
PDC [CLS] Embedding Space
1
[Doc: 2, Dist: 0.36] Req: All machine wiring must conform to NFPA 79.
2
[Doc: 2, Dist: 0.38] Req: Replacement and/or maintenance of all perishable tooling shall be designed to permit ease of removal and installation utilizing minimum tools and technical skills.
3
[Doc: 2, Dist: 0.38] Req: At all times, if feasible, all equipment shall be designed around ergonomic guidelines for improved operator comfort and performance, constructed using industry workmanship standards for operator ergonomics.

Figure 4.5: Requirement Similarity Search in PDC [CLS] Embedding Space.

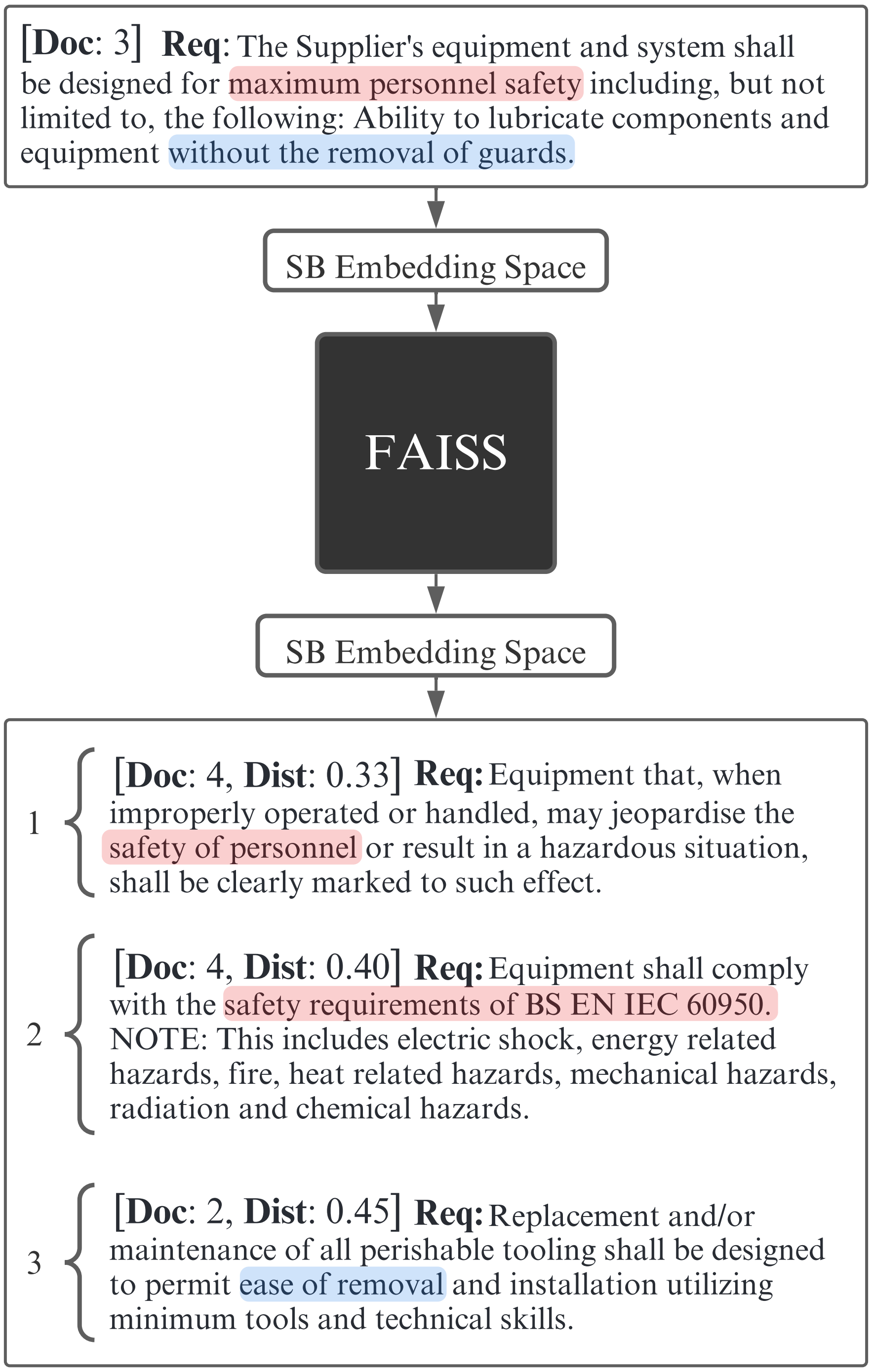

[Doc: 3] Req: The Supplier's equipment and system shall be designed for maximum personnel safety including, but not limited to, the following: Ability to lubricate components and equipment without the removal of guards.
SB Embedding Space
FAISS
SB Embedding Space
1
[Doc: 4, Dist: 0.33] Req: Equipment that, when improperly operated or handled, may jeopardise the safety of personnel or result in a hazardous situation, shall be clearly marked to such effect.
2
[Doc: 4, Dist: 0.40] Req: Equipment shall comply with the safety requirements of BS EN IEC 60950. NOTE: This includes electric shock, energy related hazards, fire, heat related hazards, mechanical hazards, radiation and chemical hazards.
3
[Doc: 2, Dist: 0.45] Req: Replacement and/or maintenance of all perishable tooling shall be designed to permit ease of removal and installation utilizing minimum tools and technical skills.

Figure 4.6: Requirement Similarity Search in SB Embedding Space.

and the second specifying a particular safety regulation that must be satisfied. The third result comes from Document 2 and was also returned in the previous search; its separation from the input requirement has increased from 0.38 to 0.45. The results as a whole have a greater range of cosine distances from the input than those shown in Figure 4.5, suggesting that the SB embeddings represent individual requirements more distinctly than the PDC [CLS] embeddings. The SB embedding results also include requirements from two documents as opposed to a single document, which may indicate a more balanced search across the entire dataset. Overall, the two embedding types return comparable results in this case, but the SB embeddings consider requirement similarity alone, while the PDC [CLS] embeddings appear to return similar requirements from similar documents.

4.3 BERT Fine-Tuned for FC

In this section, the focus shifts from the PDC task to the FC task. First, a BERT model is fine-tuned on the FC task. Table 4.5 displays training and validation metrics for each epoch of fine-tuning. As expected, the training loss decreases for each epoch as the BERT model learns the FC task. After the third epoch, however, the validation loss increases slightly, and the MCC decreases slightly. Small fluctuations like these are to be expected, but care must be taken to avoid overfitting if the number of training epochs is further increased above three.

Table 4.5: FC Training Metrics.

Epoch	Training Loss	Validation Loss	MCC
1	0.390	0.268	0.825
2	0.181	0.116	0.940
3	0.080	0.144	0.921

The confusion matrix displayed in Table 4.6 indicates the fine-tuned model's performance on the test set. As a reminder, counts for the model's predicted requirement labels are indicated along the matrix columns and counts for the actual labels are indicated along the rows. Out of the 185 NFRs, the model correctly labeled 179 for a recall of 97%; out of the 76 FRs, the model correctly labeled 63 for a recall of 83%. In terms of precision, the model receives scores of 93% for NFRs and 91% for FRs. The overall performance is summarized by an MCC of 0.82. While this MCC is lower than the PDC model's score of 0.95, it still indicates high correlation between the predicted labels and the actual labels [95]. Though the model may misclassify requirements more often than is preferable, the results clearly indicate that BERT can be fine-tuned to distinguish between FRs and NFRs.

Table 4.6: Confusion Matrix for FC BERT Model on Test Set.

		Predicted Label		
		Nonfunctional	Functional	**Totals**
Actual Label	Nonfunctional	179	6	185
	Functional	13	63	76
	Totals	192	69	261

4.3.1 Exploration of the FC Embedding Space

The [CLS] embeddings are extracted from the FC BERT model and visualized in a t-SNE plot shown in Figure 4.7. FRs and NFRs form distinct clusters, with a thin bridge connecting the two. The FR cluster could be considered two smaller clusters, while the NFRs are one sprawling cluster. To get an alternative view of the FC [CLS] embedding space, a cosine distance heatmap is shown in Figure 4.8. Once again, requirements are input beginning with Document 1's first requirement and ending with Document 5's last requirement. Bands within the heatmap reflect the observation that FRs and NFRs typically occur in groups within requirements documents.

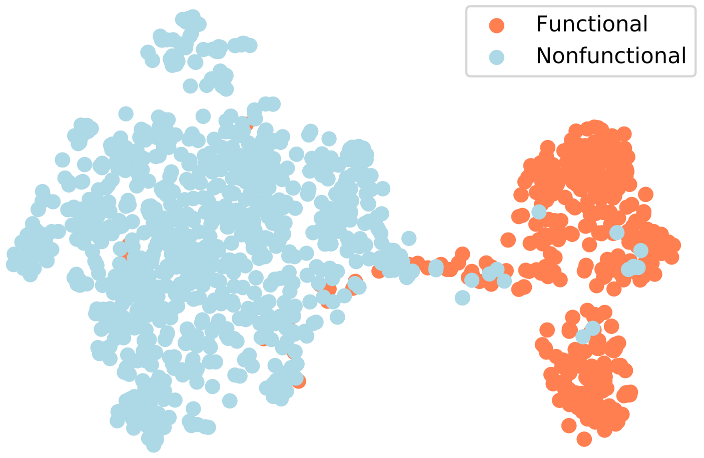


Figure 4.7: t-SNE Plot of [CLS] Embeddings Obtained from FC Task.

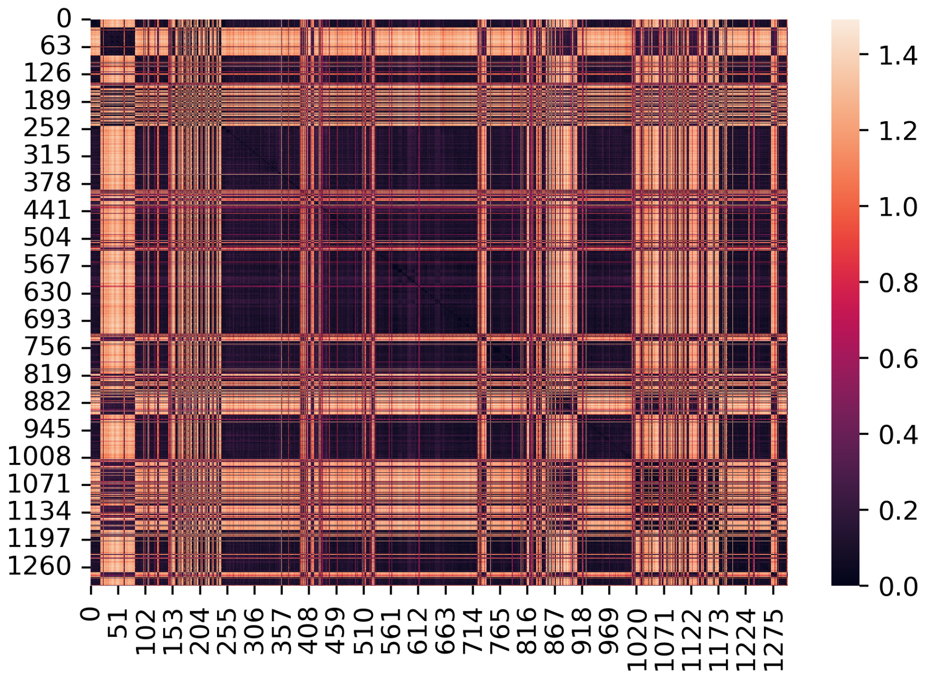


Figure 4.8: Heatmap of Cosine Distances Between FC [CLS] Embeddings.

Comparison with SB Embedding Space

For comparison, a t-SNE plot for the SB embeddings is shown in Figure 4.9. Note that this t-SNE plot is equivalent to the one shown in Figure 4.3, except the coloring is changed to indicate FRs and NFRs rather than documents. The plot shows no clear distinction between NFRs, which span the embedding space, and FRs that exist in disconnected groups. The concentration of FRs in particular areas could be due to relying on a specific definition for FRs, but not for NFRs; only requirements that describe functionality are labeled as FRs, while all other requirements are lumped together as NFRs. Overall, any recognizable requirement clusters in the SB embedding space appear more indicative of parent document relationships than functional ones.

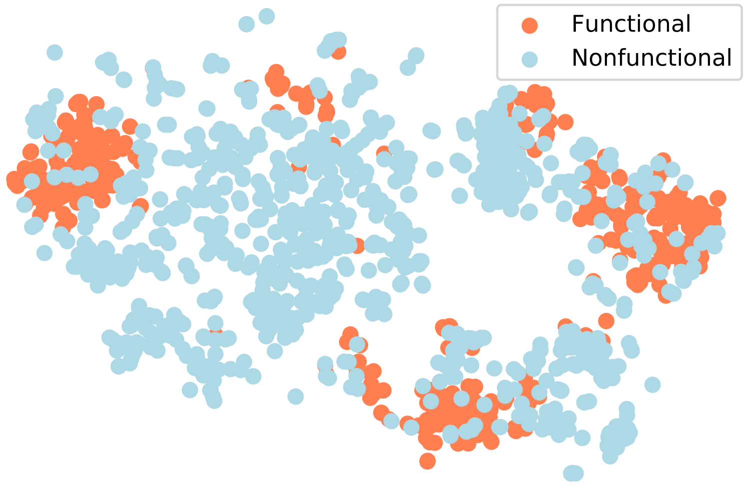


Figure 4.9: t-SNE Plot of FRs and NFRs in SB Embedding Space.

Additionally, none of the patterns shown in Figure 4.8 are especially discernible in the SB embedding heatmap previously displayed in Figure 4.4. The observations made while exploring embedding spaces indicate the SB embeddings to be more reflective of the PDC model's embeddings than the FC model's embeddings.

4.3.2 Predicting Change in Project 2

This section applies the extracted FC [CLS] embeddings to predict requirement change in Project 2. Using the embedding of a changed requirement as a query, all remaining requirements are sorted according to their embedding's cosine distance from the query; the closer a requirement is to the query, the more likely it is to experience change propagation. ECNs, shown in Table 4.7, provide documentation of requirement changes throughout the project. For the sake of brevity, specific requirements are referred to in the following sections as "R" followed by their requirement ID. The need to change a given requirement is predicted using past requirement changes. In this work, all previously changed requirements are considered change initiators, and all requirements downstream from an initiator are considered change recipients. For example, from the perspective of ECN03, R9.3.7 is a recipient and R9.2.3.1 is an initiator of change, and from the perspective of ECN04, R9.3.10 is a recipient and R9.3.7 and R9.2.3.1 are both initiators. Though initial changes do not always propagate, industry members have confirmed the ECNs presented in this work to be the result of change propagation.

Table 4.7: Project 2 Approved ECNs.

ECN	Requirements Affected
01	R9.2.3.1
03	R9.3.7
04	R9.3.10

Results are displayed in Table 4.8. Ranking refers to a recipient's position in a list of requirements that is sorted according to similarity with the initiator, and depth reflects the percentage of the sorted list that must be read before arriving at the recipient. For example, computing depth among all requirements involves dividing a ranking by Project 2's total number of requirements. All of the recipients investigated happen to be NFRs, so the results include ranking and depth among NFRs only in addition to ranking and depth among all project requirements. Project 2 contains only one FR, so ranking and depth remain nearly equivalent whether the FR is omitted or not. The row of the initiator that best predicts a recipient is highlighted for clarity. As the only initiator that can be used to predict R9.3.7, R9.2.3.1 yields a ranking of 33 and depth of 21%. R9.2.3.1 also best predicts change propagation to R9.3.10, with a ranking of 24 and depth of 15%.

Table 4.8: Project 2 Change Prediction Results with FC [CLS] Embeddings.

Recipient	Initiator	Among All Requirements		Among NFRs	
		Ranking	Depth	Ranking	Depth
R9.3.7	R9.2.3.1	33	21%	33	21%
R9.3.10	R9.2.3.1	24	15%	23	15%
	R9.3.7	51	32%	50	32%

Comparison to SB Results

For comparison, Table 4.9 presents the SB embeddings' change prediction results. The SB embeddings achieve overall higher rankings and depths ("higher" meaning smaller in magnitude and therefore more desirable for change prediction), indicating a greater aptitude for change prediction than the FC [CLS] embeddings. Most notably, the SB embeddings perfectly rank R9.3.10 as the most likely recipient of change initiated by R9.3.7. This contrasts with the FC [CLS] embedding results that instead claim R9.3.10 to have a stronger relationship with R9.2.3.1, with a ranking of 24. Further, Table 4.9 indicates that SB embeddings predict both recipients with a depth of at most 10% for all initiators, while the lowest depth achieved by the FC [CLS] embeddings is 15%. In comparing results from both sets of embeddings, it becomes clear that the SB embeddings are preferable to the FC [CLS] embeddings for predicting requirement change in Project 2. The presumed strength of the FC [CLS] embeddings is their ability to consider functionality while predicting change; this feature is of little use in Project 2, which contains a single FR.

Table 4.9: Project 2 Change Prediction Results with SB Embeddings.

Recipient	Initiator	Among All Requirements		Among NFRs	
		Ranking	Depth	Ranking	Depth
R9.3.7	R9.2.3.1	15	9%	15	9%
R9.3.10	R9.2.3.1	16	10%	15	9%
	R9.3.7	1	< 1%	1	< 1%

4.3.3 Predicting Change in Project 3

The application of requirement embeddings for change prediction is also explored in Project 3. Composed of 20% FRs and 80% NFRs, Project 3 is more balanced than Project 2, so it should reveal the merit of considering functionality while predicting change. Table 4.10 displays the analyzed Project 3 ECNs.

Table 4.10: Project 3 Approved ECNs.

ECN	Requirements Affected
01	R2.5.8 – R2.1.2 – R2.9.2 – R2.1.14
07	R2.1.14 – R2.2.6
11	R2.7

The change prediction results obtained with the FC [CLS] embeddings are displayed in Table 4.11. Note that R2.1.14 is ignored as a recipient in ECN07 since it is also one of the change initiators affected by ECN01. Of the five initiators in ECN01, R2.5.8 is found to be the best predictor of downstream change in the recipient, R2.2.6, with a depth of 26% among all requirements and 31% among NFRs. The other analyzed recipient, R2.7, is best predicted by R2.9.2, with a depth of 49% among all requirements and 58% among NFRs. For each result, there are improvements in ranking when analyzed among NFRs, but search depth among NFRs increases compared to search depth among all requirements. Removing FRs has only a minor impact on ranking while reducing the list of considered requirements by 42. Maintaining a nearly equivalent ranking in a shorter list produces the observed increase in search depth.

Table 4.11: Project 3 Change Prediction Results with FC [CLS] Embeddings.

Recipient	Initiator	Among All Requirements		Among NFRs	
		Ranking	Depth	Ranking	Depth
R2.2.6	R2.5.8	56	26%	53	31%
	R2.1.2	76	36%	76	44%
	R2.9.2	137	63%	131	76%
	R2.1.14	79	37%	76	44%
R2.7	R2.5.8	134	63%	129	75%
	R2.1.2	157	73%	150	87%
	R2.9.2	105	49%	99	58%
	R2.1.14	151	71%	144	84%
	R2.2.6	156	73%	151	88%

Comparison to SB Results

Once again, the change prediction results generated with the FC [CLS] embeddings are compared to those produced by the SB embeddings, shown in Table 4.12. Each recipient is best predicted by a different initiator from those shown in Table 4.11. With a nearly perfect ranking and search depth of 1% and 2% among all requirements and NFRs, respectively, recipient R2.2.6 is best predicted by R2.1.14. Predicting recipient R2.7 proves to be a greater challenge, with R2.5.8 yielding the best depths of 31% among all requirements and 30% among NFRs. Though the prediction of R2.7 is the least successful of all SB embedding predictions in Projects 2 and 3, it still remains better than the FC [CLS] embedding prediction of R2.7.

Table 4.12: Project 3 Change Prediction Results with SB Embeddings.

Recipient	Initiator	Among All Requirements		Among NFRs	
		Ranking	Depth	Ranking	Depth
R2.2.6	R2.5.8	79	37%	62	36%
	R2.1.2	13	6%	12	7%
	R2.9.2	72	34%	47	27%
	R2.1.14	3	1%	3	2%
R2.7	R2.5.8	66	31%	52	30%
	R2.1.2	128	60%	99	58%
	R2.9.2	90	42%	62	36%
	R2.1.14	86	40%	64	37%
	R2.2.6	97	45%	76	44%

The SB embedding results exhibit the opposite pattern of the FC [CLS] embedding results: search depth among NFRs is frequently less than search depth among all requirements. Removing FRs from the list of considered requirements can be interpreted as eliminating "noise" from the change prediction search. An improvement in ranking after removing FRs is to be expected, as at least some FRs are likely to be ranked ahead of the recipient. The improvement in search depth, however, signifies that when considering all requirements, enough FRs were ranked ahead of the recipient to obscure its ranking with respect to other NFRs. That is, removing FRs from Project 3's change prediction search improves the relevance of results, albeit by a small amount. Designers may benefit from separating FRs and NFRs before conducting a change prediction search to capture the most relevant results. If it is evident that change could only propagate to either FRs or NFRs, as is the case in Projects 2 and 3, then only those requirements should be included in the change prediction search.

Overall, the SB embeddings clearly outperform the FC [CLS] embeddings when applied to change prediction in Projects 2 and 3. However, the noted improvement in search depth among NFRs suggests that a combination of the two models may be ideal: first, requirements could automatically be classified as FRs or NFRs with the FC BERT model, and then SB embeddings could predict change in either set of requirements to yield the most relevant results.

Chapter 5

Discussion

5.1 Addressing Research Questions

Having completed each step in the research map presented in Figure 1.1, the acquired results and observations are used to answer this work's three research questions.

RQ1: Can BERT's requirement embeddings differentiate across requirements documents?

Based on the MCC of 0.95 achieved on the PDC task, this work demonstrates that BERT can indeed differentiate across requirements documents. By obtaining a nearly perfect MCC, BERT proves its ability to identify the nuances that distinguish and relate requirements documents. Even though its pre-training corpus does not contain requirements, transfer learning allows BERT to be fine-tuned to recognize the semantic and syntactic patterns specific to requirements documents.

RQ2: Can BERT's requirement embeddings differentiate between FRs and NFRs?

With an MCC of 0.82 computed for the FC task, this work indicates that BERT can also differentiate between FRs and NFRs, further demonstrating the level of detail contained in BERT's requirement representations. Not only can BERT recognize inter-document requirement patterns, but also intra-document requirement patterns. The successful classification results at these two levels of granularity suggest that BERT is suitable for a broad range of requirements management applications.

RQ3: Do embeddings from BERT fine-tuned on requirement classification tasks yield better results in requirements management applications than general-language embeddings?

This research question does not have a clear yes or no answer; the relative performance of BERT's fine-tuned embeddings depends on both the fine-tuning task and the particular requirements management application. Therefore, this question is addressed by considering how the PDC and FC models' [CLS] embeddings compare to the SB embeddings in each of the presented applications.

When computing similarity at the document level, the PDC [CLS] embeddings appear to have an advantage over the SB embeddings. The PDC [CLS] embeddings represent each document distinctly and exhibit clearly recognizable patterns between them, only some of which are reflected in the SB embeddings. Conversely, the SB embeddings are better equipped to perform requirement similarity searches, as they represent individual requirements distinctly and search across the entire dataset, whereas the PDC [CLS] embeddings tend to search only among the document closest to the query requirement's document.

In every explored instance of change prediction, the SB embeddings yielded better predictions than the FC [CLS] embeddings. As noted at the end of Section 4.3.3, however, in some cases the most relevant results are produced by using the two models in tandem; the FC BERT model groups requirements into FRs and NFRs, and then a similarity search is performed within a group via SB embeddings. It should be noted that this approach relies on designers to anticipate which group, either FRs or NFRs, is most likely to be affected by an initial change.

In summary, the embedding performance depends on how well the fine-tuning task relates to the requirements management application. For instance, the SB embeddings are generated by a model that is fine-tuned specifically to represent general-language sequences for similarity searches, while the PDC and FC [CLS] embeddings are extracted from models that are fined-tuned only to recognize certain types of similarity, i.e., similarity based on parent document and similarity based on functionality. Consequently, the SB embeddings yield superior performance in both applications that involve a similarity search among requirements. The SB embeddings fail to outperform the PDC [CLS] embeddings when computing document similarities, however, because the PDC task specifically requires the BERT model to learn embeddings that distinguish documents. This work's recommendation is to use general-language sequence embeddings, such as those produced by the SB model, in requirements management applications unless sufficient labeled data exists to fine-tune a model for the exact desired application.

5.2 Impact on Design Research and Practice

With respect to design research, this work reaffirms the findings of previous studies that have applied transformer-based models to analyze design requirements. Having verified such models' ability to create detailed requirement embeddings, this work motivates future studies to further explore fine-tuning on requirements management tasks. Currently, one of the greatest challenges for researchers in computational design is the lack of publicly available, labeled requirements datasets. Aside from intellectual contributions, this work contributes to the design research community by releasing a portion of its labeled dataset. Though the remaining documents used in this work cannot be released, Documents 4 and 5 (580 requirements combined) labeled for the FC task are available on GitHub[1].

This work's potential impact on design practice is envisioned through a theoretical scenario. Recall the period in the COVID-19 pandemic when there was a shortage of ventilators. Several companies with little relevant experience expressed a desire to begin producing ventilators. Such a company would benefit from determining which of its previously completed projects have the greatest overlap with ventilator design. After analyzing the design documentation, a transformer-based model could promptly rank projects according to similarity, allowing the company to get a quick start based on existing work. Once the company begins designing, their inexperience manifests in the need for many ECNs throughout the design process. Rather than repeatedly scanning through an entire requirements document for potential change propagation paths, a designer could use a transformer-based model to generate a sorted list of most likely change recipients, allowing more time to make the necessary preparations for any potential future change. Though idealized, this scenario provides a glimpse of how automated requirements management tools

[1]https://github.com/jessemullis/SKA_docs_ForNF

built on transformer-based models could empower designers to maximize their agility while minimizing unnecessary costs and delays.

5.3 Limitations

Several limitations must be addressed to put this work's contributions into context. First, this work's dataset is not verified to be representative of all system requirements. Additionally, the dataset is large when compared to available collections of system requirements, but remains dwarfed by other fine-tuning datasets. For instance, the SB model is fined-tuned on 215M examples. Increasing this work's dataset by at least an order of magnitude could improve generalizability. The PDC and FC tasks also vary in their potential to create generalizable models. A model trained on the PDC task is only applicable to documents included in its training dataset; the model's task is to classify requirements according to the documents it was trained on and those documents only. Since all requirements can be classified as FRs or NFRs, models trained on the FC task can generalize, but only if definitions of FRs and NFRs remain consistent. Finally, it may prove difficult to exactly replicate this work's results, even with the same dataset. Prior to fine-tuning, some of BERT's parameters are randomly initialized and there is no guarantee that models will approach the same optimum. Several models were fine-tuned throughout this work and each yielded slightly different results, though the general patterns remained consistent.

Chapter 6

Conclusion and Future Work

The immense potential for NLP in requirements management applications has long been established. Yet, no previous work has determined whether state-of-the-art NLP models, like BERT, can sufficiently represent and distinguish requirements. Using a diverse set of 1,303 requirements sourced from five system design projects, this work confirms that BERT can differentiate between requirements according to both parent document and functionality. After fine-tuning BERT on the PDC and FC tasks, the models' requirement-specific [CLS] embeddings are compared to general-language SB embeddings in requirements management applications. Namely, this work applies requirement embeddings to compute similarity and predict change. The PDC [CLS] embeddings outperform the SB embeddings for identifying document-level similarity, but the SB embeddings are superior for requirement-level similarity searches and change prediction.

This work supports the development of automated requirements management tools that rely on NLP models. With NLP's rapid progression in recent years, language embeddings have become increasingly capable of automatically revealing relationships between documents and requirements. These relationships can be unexpected and may guide designers to uncover patterns that would be missed otherwise. Though similarity and change prediction searches may not always produce relevant results, automated requirements management tools are not meant to replace humans entirely; rather, these tools' purpose is to provide suggestions that can be pursued or ignored at a designer's discretion.

Two branches of future work are suggested: one that investigates requirement embeddings and one that explores their applications in requirements management. To further examine requirement embeddings, future work could compare embeddings from each of BERT's layers, as it remains unclear which is most informative. Additionally, BERT is only one of many transformer-based models, so future work may involve comparing BERT's requirement embeddings with those generated by alternative models, such as MPNet [96]. To further investigate requirements management applications, a study that gathers human evaluations of requirement similarity within the dataset would provide a baseline for judging a model's requirement similarity analysis. While this work compared whole documents by averaging their requirements' embeddings, other methods of representing documents may be preferable. Finally, the SB embeddings yielded promising change prediction results that future work could build upon by considering the joint effect of multiple change initiators.

Bibliography

[1] Pasquale Lops, Marco de Gemmis, and Giovanni Semeraro. “Content-based Recommender Systems: State of the Art and Trends”. In: *Recommender Systems Handbook*. Boston, MA: Springer US, 2011, pp. 73–105. ISBN: 978-0-387-85820-3. DOI: 10.1007/978-0-387-85820-3_3. URL: https://doi.org/10.1007/978-0-387-85820-3_3.

[2] Nouh T. Alhindawi. “Information Retrieval - Based Solution for Software Requirements Classification and Mapping”. In: *2018 5th International Conference on Mathematics and Computers in Sciences and Industry (MCSI)*. 2018, pp. 147–154. DOI: 10.1109/MCSI.2018.00042.

[3] Suresh Yadla, Jane Huffman Hayes, and Alex Dekhtyar. “Tracing requirements to defect reports: an application of information retrieval techniques”. In: *Innovations in Systems and Software Engineering* 1.2 (2005), pp. 116–124. ISSN: 1614-5046. DOI: 10.1007/s11334-005-0011-3.

[4] Han Hu et al. “Toward Scalable Systems for Big Data Analytics: A Technology Tutorial”. In: *IEEE Access* 2 (2014), pp. 652–687. DOI: 10.1109/ACCESS.2014.2332453.

[5] Dnyanesh Rajpathak, Prakash M. Peranandam, and S. Ramesh. “Automatic development of requirement linking matrix based on semantic similarity for robust software development”. In: *Journal of Systems and Software* 186 (2022), p. 111211. ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2021.111211. URL: https://www.sciencedirect.com/science/article/pii/S016412122100282X.

[6] Fatma A. Mihany et al. “An Automated System for Measuring Similarity between Software Requirements”. In: *Proceedings of the 2nd Africa and Middle East Conference on Software Engineering*. AMECSE ’16. Cairo, Egypt: Association for Computing Machinery, 2016, pp. 46–51. ISBN: 9781450342933. DOI: 10.1145/2944165.2944173. URL: https://doi.org/10.1145/2944165.2944173.

[7] Kareshna Zamani. “A Prediction Model for Software Requirements Change Impact”. In: *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 2021, pp. 1028–1032. DOI: 10.1109/ASE51524.2021.9678582.

[8] G. Pahl and W. Beitz. *Engineering design : a systematic approach*. 3rd ed. London : Springer, 2007. ISBN: 9781846283185.

[9] Charles W. Krueger. “Easing the Transition to Software Mass Customization”. In: *Revised Papers from the 4th International Workshop on Software Product-Family Engineering*. PFE ’01. Berlin, Heidelberg: Springer-Verlag, 2001, pp. 282–293. ISBN: 3540436596.

[10] Naveed Ahmad, David Wynn, and John P. Clarkson. “Change impact on a product and its redesign process: A tool for knowledge capture and reuse”. In: *Research in Engineering Design* (July 2012), pp. 1–26. DOI: 10.1007/s00163-012-0139-8.

[11] Beshoy Morkos, James Mathieson, and Joshua Summers. “Comparative analysis of requirements change prediction models: Manual, linguistic, and neural network”. In: *Research in Engineering Design* 25 (Apr. 2014), pp. 139–156. DOI: 10.1007/s00163-014-0170-z.

[12] Yang Li and Tao Yang. “Word Embedding for Understanding Natural Language: A Survey”. In: *Guide to Big Data Applications*. Cham: Springer International Publishing, 2018, pp. 83–104. ISBN: 978-3-319-53817-4. DOI: 10.1007/978-3-319-53817-4_4. URL: https://doi.org/10.1007/978-3-319-53817-4_4.

[13] Tobias Hey et al. “NoRBERT: Transfer Learning for Requirements Classification”. In: *2020 IEEE 28th International Requirements Engineering Conference (RE)*. 2020, pp. 169–179. DOI: 10.1109/RE48521.2020.00028.

[14] Haluk Akay and Sang-Gook Kim. “Measuring functional independence in design with deep-learning language representation models”. In: *Procedia CIRP* 91 (2020), pp. 528–533. DOI: 10.1016/j.procir.2020.02.210.

[15] Jinfeng Lin et al. “Traceability Transformed: Generating More Accurate Links with Pre-Trained BERT Models”. In: *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. May 2021, pp. 324–335. DOI: 10.1109/ICSE43902.2021.00040.

[16] Muhammad Abbas et al. “On the relationship between similar requirements and similar software”. In: *Requirements Engineering* (2022). ISSN: 1432-010X. DOI: 10.1007/s00766-021-00370-4. URL: https://doi.org/10.1007/s00766-021-00370-4.

[17] Beshoy Morkos, Prabhu Shankar, and Joshua D. Summers. “Predicting requirement change propagation, using higher order design structure matrices: an industry case study”. In: *Journal of Engineering Design* 23.12 (2012), pp. 905–926. DOI: 10.1080/09544828.2012.662273.

[18] *Requirement Modeling Systems for Mechanical Design: A Systematic Method for Evaluating Requirement Management Tools and Languages*. Vol. Volume 3: 30th Computers and Information in Engineering Conference, Parts A and B. International Design Engineering Technical Conferences and Computers and Information in Engineering Conference. Aug. 2010, pp. 1247–1257. DOI: 10.1115/DETC2010-28989. URL: https://doi.org/10.1115/DETC2010-28989.

[19] *Requirements and Data Content Evaluation of Industry In-House Data Management System*. Vol. Volume 3: 30th Computers and Information in Engineering Conference, Parts A and B. International Design Engineering Technical Conferences and Computers and Information in Engineering Conference. 2010, pp. 493–503. DOI: 10.1115/DETC2010-28548. URL: https://doi.org/10.1115/DETC2010-28548.

[20] *Requirement Change Propagation Prediction Approach: Results From an Industry Case Study*. Vol. Volume 1: 36th Design Automation Conference, Parts A and B. International Design Engineering Technical Conferences and Computers and Information in Engineering Conference. 2010, pp. 111–121. DOI: 10.1115/DETC2010-28562. eprint: https://asmedigitalcollection.asme.org/IDETC-CIE/proceedings-pdf/IDETC-CIE2010/44090/111/2696687/111_1.pdf. URL: https://doi.org/10.1115/DETC2010-28562.

[21] Beshoy Morkos, Shraddha Joshi, and Joshua D. Summers. "Representation: Formal Development and Computational Recognition of Localized Requirement Change Types". In: *ASME International Design Engineering Technical Conference* 3 (2012). DOI: 10.1115/DETC2012-71417.

[22] *Exploring Requirement Change Propagation Through the Physical and Functional Domain*. Vol. Volume 1B: 35th Computers and Information in Engineering Conference. International Design Engineering Technical Conferences and Computers and Information in Engineering Conference. Aug. 2015. DOI: 10.1115/DETC2015-47746. URL: https://doi.org/10.1115/DETC2015-47746.

[23] Phyo Htet Hein, Nathaniel Voris, and Beshoy Morkos. "Predicting Requirement Change Propagation Through Investigation of Physical and Functional Domains". In: *Research in Engineering Design* 29.2 (2018), pp. 309–328. DOI: 10.1007/s00163-017-0271-6.

[24] Phyo Htet Hein, Beshoy Morkos, and Chiradeep Sen. "Utilizing Node Interference Method and Complex Network Centrality Metrics to Explore Requirement Change Propagation". In: *Proceedings of the ASME International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, 2017, VOL 1*. 2017. ISBN: 978-0-7918-5811-0.

[25] Joshua Summers, Shraddha Joshi, and Beshoy Morkos. "Requirements Evolution: Relating Functional and Non-Functional Requirement Change on Student Project Success". In: Aug. 2014, V003T04A002. DOI: 10.1115/DETC2014-35023.

[26] Beshoy Morkos, Shraddha Joshi, and Joshua D. Summers. "Investigating the impact of requirements elicitation and evolution on course performance in a pre-capstone design course". In: *Journal of Engineering Design* 30.4-5 (2019), pp. 155–179. ISSN: 0954-4828. DOI: 10.1080/09544828.2019.1605584.

[27] Shraddha Joshi, Beshoy Morkos, and Joshua D. Summers. "Mapping problem and requirements to final solution: A document analysis of capstone design projects". In: *International Journal of Mechanical Engineering Education* 47.4 (2019), pp. 338–370. DOI: 10.1177/0306419018780741. URL: https://doi.org/10.1177/0306419018780741.

[28] Prabhu Shankar, Beshoy Morkos, and Joshua D. Summers. "Reasons for change propagation: a case study in an automotive OEM". In: *Research in Engineering Design* 23.4, SI (2012), pp. 291–303. ISSN: 0934-9839. DOI: 10.1007/s00163-012-0132-2.

[29] Phyo Htet Hein et al. "Employing machine learning techniques to assess requirement change volatility". In: *Research in Engineering Design* 32.2 (2021), pp. 245–269. DOI: 10.1007/s00163-020-00353-6.

[30] Prabhu Shankar et al. "Towards the formalization of non-functional requirements in conceptual design". In: *Research in Engineering Design* 31 (Oct. 2020), pp. 449–469. DOI: 10.1007/s00163-020-00345-6.

[31] Cheng Chen, Jesse Mullis, and Beshoy Morkos. "A Topic Modeling Approach to Study Design Requirements". In: vol. Volume 3A: 47th Design Automation Conference (DAC). International Design Engineering Technical Conferences and Computers and Information in Engineering Conference. 2021. DOI: 10.1115/DETC2021-72151.

[32] Daniel Long, Beshoy Morkos, and Scott Ferguson. "Toward Quantifiable Evidence of Excess' Value Using Personal Gaming Desktops". In: *Journal of Mechanical Design* 143.3 (2021). ISSN: 1050-0472. DOI: 10.1115/1.4049520. URL: https://doi.org/10.1115/1.4049520.

[33] *Forecasting the Value of Excess in Personal Gaming Desktops*. Vol. Volume 11B: 46th Design Automation Conference (DAC). International Design Engineering Technical Conferences and Computers and Information in Engineering Conference. 2020. DOI: 10.1115/DETC2020-22248. URL: https://doi.org/10.1115/DETC2020-22248.

[34] *Estimating the Value of Excess: A Case Study of Gaming Computers, Consoles and the Video Game Industry*. Vol. Volume 2A: 45th Design Automation Conference. International Design Engineering Technical Conferences and Computers and Information in Engineering Conference. 2019. DOI: 10.1115/DETC2019-98428. URL: https://doi.org/10.1115/DETC2019-98428.

[35] Karl T Ulrich and Steven D. Eppinger. *Product design and development*. eng. 2nd ed. Boston: Irwin/McGraw-Hill, 2000. ISBN: 007229647X.

[36] Harold Halbleib. "Requirements Management". In: *Information Systems Management* 21.1 (2004), pp. 8–14.

[37] Jacob Shabi et al. "A decision support model to manage overspecification in system development projects". In: *Journal of Engineering Design* 32.7 (2021), pp. 323–345.

[38] Karl Wiegers. *Software Requierements : Practical Techniques for Gathering and Managing Requirements Throughout the Product Development Cycle] K.E. Wiegers.* Microsoft Press, 2003.

[39] Gerald Kotonya and Ian Sommerville. *Requirements engineering: processes and techniques*. John Wiley & Sons, Inc., 1998.

[40] A. Terry Bahill and Steven J. Henderson. “Requirements development, verification, and validation exhibited in famous failures”. In: *Systems engineering : the journal of the International Council on Systems Engineering.* 8.1 (2005), p. 1.

[41] Massila Kamalrudin, Nuridawati Mustafa, and Safiah Sidek. “A Template for Writing Security Requirements”. In: *Requirements Engineering for Internet of Things*. Springer Singapore, 2018, pp. 73–86.

[42] Anne Condamines and Maxime Warnier. “Towards the creation of a CNL adapted to requirements writing by combining writing recommendations and spontaneous regularities: example in a space project”. In: *Language Resources and Evaluation* 51.1 (2017), pp. 221–247.

[43] Robert Shishko. *NASA Systems Engineering Handbook*. National Aeronautics and Space Administration, 1995.

[44] International Council on Systems Engineering. *Guide for Writing Requirements*. 2019.

[45] Aerospace and Defense Industries Association of Europe. *Simplified Technical English*. 2021.

[46] PMI. “Requirements Management: Core Competency for Project and Program Success”. In: (2014).

[47] John R. Hauser and Don Clausing. “House of Quality”. In: *Harvard Business Review* (1988).

[48] Lawrence Chung and Julio Cesar Sampaio do Prado Leite. “On Non-Functional Requirements in Software Engineering”. In: *Conceptual Modeling: Foundations and Applications: Essays in Honor of John Mylopoulos*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 363–379. ISBN: 978-3-642-02463-4. DOI: `10.1007/978-3-642-02463-4_19`. URL: `https://doi.org/10.1007/978-3-642-02463-4_19`.

[49] Lawrence Chung, Julio Cesar, and Sampaio Prado Leite. *Non-functional requirements in software engineering*. 1999.

[50] Zijad Kurtanovic and Walid Maalej. “Automatically Classifying Functional and Non-functional Requirements Using Supervised Machine Learning”. In: *2017 IEEE 25th International Requirements Engineering Conference (RE)*. 2017, pp. 490–495. DOI: `10.1109/RE.2017.82`.

[51] B Caglayan et al. *The PROMISE repository of empirical software engineering data*. Jan. 2012.

[52] Raul Navarro-Almanza, Reyes Juarez-Ramirez, and Guillermo Licea. “Towards Supporting Software Engineering Using Deep Learning: A Case of Software Requirements Classification”. In: *2017 5th International Conference in Software Engineering Research and Innovation (CONISOFT)*. 2017, pp. 116–120. DOI: `10.1109/CONISOFT.2017.00021`.

[53] Jonas Winkler and Andreas Vogelsang. “Automatic Classification of Requirements Based on Convolutional Neural Networks”. In: *2016 IEEE 24th International Requirements Engineering Conference Workshops (REW)*. 2016, pp. 39–45. DOI: `10.1109/REW.2016.021`.

[54] Alec Radford and Karthik Narasimhan. “Improving Language Understanding by Generative Pre-Training”. In: (2018).

[55] Haluk Akay and Sang-Gook Kim. "Design transcription: Deep learning based design feature representation". In: *CIRP Annals* 69.1 (2020), pp. 141–144. DOI: https://doi.org/10.1016/j.cirp.2020.04.084.

[56] Sebastian Schubert, Arun Nagarajah, and Joerg Feldhusen. "An Approach for More Efficient Variant Design Processes". In: *Proceedings of the 18th International Conference on Engineering Design (ICED 11), Impacting Society Through Engineering Design*. Ed. by S Culley et al. Vol. 4. 2011, pp. 157–166. ISBN: 978-1-904670-24-7.

[57] Qin Yang et al. "Configuration Equilibrium Model of Product Variant Design Driven by Customer Requirements". In: *Symmetry* 11.4 (2019). ISSN: 2073-8994. DOI: 10.3390/sym11040508. URL: https://www.mdpi.com/2073-8994/11/4/508.

[58] J.H. Hayes, A. Dekhtyar, and S.K. Sundaram. "Advancing candidate link generation for requirements tracing: the study of methods". In: *IEEE Transactions on Software Engineering* 32.1 (2006), pp. 4–19. DOI: 10.1109/TSE.2006.3.

[59] Sebastian Eder et al. "Configuring Latent Semantic Indexing for Requirements Tracing". In: *Proceedings of the Second International Workshop on Requirements Engineering and Testing*. Florence, Italy: IEEE Press, 2015, pp. 27–33.

[60] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-term Memory". In: *Neural computation* 9 (Dec. 1997), pp. 1735–80. DOI: 10.1162/neco.1997.9.8.1735.

[61] KyungHyun Cho et al. "On the Properties of Neural Machine Translation: Encoder-Decoder Approaches". In: *CoRR* (2014). URL: http://arxiv.org/abs/1409.1259.

[62] Jin Guo, Jinghui Cheng, and Jane Cleland-Huang. "Semantically Enhanced Software Traceability Using Deep Learning Techniques". In: *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. 2017, pp. 3–14. DOI: 10.1109/ICSE.2017.9.

[63] Alessio Ferrari, Giorgio Oronzo Spagnolo, and Stefania Gnesi. "PURE: A Dataset of Public Requirements Documents". In: *2017 IEEE 25th International Requirements Engineering Conference (RE)*. 2017, pp. 502–505. DOI: 10.1109/RE.2017.29.

[64] Timothy Jarratt, John Clarkson, and Claudia Eckert. "Engineering change". In: *Design process improvement: A review of current practice*. London: Springer London, 2005, pp. 262–285. DOI: 10.1007/978-1-84628-061-0_11.

[65] Claudia Eckert, P. John Clarkson, and Winfried Zanker. "Change and customisation in complex engineering domains". In: *Research in Engineering Design* 15.1 (2004), pp. 1–21. DOI: 10.1007/s00163-003-0031-7.

[66] P. John Clarkson, Caroline Simons, and Claudia Eckert. "Predicting Change Propagation in Complex Design". In: *Journal of Mechanical Design* 126.5 (2004), pp. 788–797. DOI: 10.1115/1.1765117.

[67] Jihwan Lee and Yoo S. Hong. “Bayesian network approach to change propagation analysis”. In: *Research in Engineering Design* 28.4 (2017), pp. 437–455. DOI: 10.1007/s00163-017-0252-9.

[68] Thorsten Brants and Alex Franz. *Web 1T 5-gram*. 2006.

[69] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge, UK: Cambridge University Press, 2008.

[70] Lajanugen Logeswaran and Honglak Lee. “An efficient framework for learning sentence representations”. In: *International Conference on Learning Representations*. 2018.

[71] Yizhe Zhang et al. “Deconvolutional Paragraph Representation Learning”. In: *NIPS*. 2017.

[72] Minmin Chen. “Efficient Vector Representation for Documents through Corruption”. In: *ArXiv* abs/1707.02377 (2017).

[73] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. “Linguistic Regularities in Continuous Space Word Representations”. In: *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, 2013, pp. 746–751.

[74] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. “Learning representations by back-propagating errors”. In: *Nature* 323 (1986), pp. 533–536.

[75] Yoshua Bengio et al. “A Neural Probabilistic Language Model”. In: *J. Mach. Learn. Res.* 3 (2003), pp. 1137–1155. ISSN: 1532-4435.

[76] Tomas Mikolov et al. “Efficient Estimation of Word Representations in Vector Space”. In: *Proceedings of Workshop at ICLR* (2013).

[77] Jeffrey Pennington, Richard Socher, and Christopher Manning. “GloVe: Global Vectors for Word Representation”. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2014, pp. 1532–1543. DOI: 10.3115/v1/D14-1162.

[78] Ashish Vaswani et al. “Attention Is All You Need”. In: (2017). arXiv: 1706.03762 [cs.CL].

[79] Matthew E. Peters et al. “Deep contextualized word representations”. In: *CoRR* (2018). arXiv: 1802.05365. URL: http://arxiv.org/abs/1802.05365.

[80] J. Devlin et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *NAACL*. 2019.

[81] Sinno Jialin Pan and Qiang Yang. “A Survey on Transfer Learning”. In: *IEEE Transactions on Knowledge and Data Engineering* 22.10 (2010), pp. 1345–1359. DOI: 10.1109/TKDE.2009.191.

[82] Wilson L. Taylor. ““Cloze Procedure”: A New Tool for Measuring Readability”. In: *Journalism Quarterly* 30.4 (1953), pp. 415–433. DOI: 10.1177/107769905303000401. eprint: https://doi.org/10.1177/107769905303000401. URL: https://doi.org/10.1177/107769905303000401.

[83] Yonghui Wu et al. "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation". In: *CoRR* abs/1609.08144 (2016). arXiv: `1609.08144`. URL: `http://arxiv.org/abs/1609.08144`.

[84] Pranav Rajpurkar et al. "SQuAD: 100,000+ Questions for Machine Comprehension of Text". In: *arXiv e-prints* (2016). eprint: `1606.05250`.

[85] Jacob Cohen. "A Coefficient of Agreement for Nominal Scales". In: *Educational and Psychological Measurement* 20.1 (1960), pp. 37–46. DOI: `10.1177/001316446002000104`. URL: `https://doi.org/10.1177/001316446002000104`.

[86] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[87] Mary McHugh. "Interrater reliability: The kappa statistic". In: *Biochemia medica : časopis Hrvatskoga društva medicinskih biokemičara / HDMB* 22 (Oct. 2012), pp. 276–82. DOI: `10.11613/BM.2012.031`.

[88] Thomas Wolf et al. "Transformers: State-of-the-Art Natural Language Processing". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online: Association for Computational Linguistics, 2020, pp. 38–45. DOI: `10.18653/v1/2020.emnlp-demos.6`. URL: `https://aclanthology.org/2020.emnlp-demos.6`.

[89] Giuseppe Jurman, Samantha Riccadonna, and Cesare Furlanello. "A Comparison of MCC and CEN Error Measures in Multi-Class Prediction". In: *PLOS ONE* 7.8 (Aug. 2012), pp. 1–8. DOI: `10.1371/journal.pone.0041882`. URL: `https://doi.org/10.1371/journal.pone.0041882`.

[90] Davide Chicco, Niklas Tötsch, and Giuseppe Jurman. "The Matthews correlation coefficient (MCC) is more reliable than balanced accuracy, bookmaker informedness, and markedness in two-class confusion matrix evaluation". In: *BioData Mining* 14 (Feb. 2021). DOI: `10.1186/s13040-021-00244-z`.

[91] Laurens Van der Maaten and Geoffrey Hinton. "Visualizing Data using t-SNE". In: *Journal of Machine Learning Research* 9.86 (2008), pp. 2579–2605. URL: `http://jmlr.org/papers/v9/vandermaaten08a.html`.

[92] Nils Reimers and Iryna Gurevych. "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2019. URL: `https://arxiv.org/abs/1908.10084`.

[93] Charu C. Aggarwal, Alexander Hinneburg, and Daniel A. Keim. "On the Surprising Behavior of Distance Metrics in High Dimensional Space". In: *Database Theory — ICDT 2001*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 420–434. ISBN: 978-3-540-44503-6.

[94] Jeff Johnson, Matthijs Douze, and Hervé Jégou. "Billion-scale similarity search with GPUs". In: *arXiv preprint arXiv:1702.08734* (2017).

[95] Dennis E. Hinkle, William Wiersma, and Stephen G. Jurs. *Applied statistics for the behavioral sciences*. eng. 3rd ed. Boston: Houghton Mifflin, 1994. ISBN: 0395675553.

[96] Kaitao Song et al. “MPNet: Masked and Permuted Pre-training for Language Understanding”. In: *CoRR* abs/2004.09297 (2020). URL: `https://arxiv.org/abs/2004.09297`.

Appendix A

Code for Fine-tuning Bert on PDC Task

```
# Make sure GPU is equipped
gpu_info = !nvidia-smi
gpu_info = '\n'.join(gpu_info)
if gpu_info.find('failed') >= 0:
  print('Not connected to a GPU')
else:
  print(gpu_info)
```

```
# Install transformers package in colab session
!pip install transformers
```

```
# Link colab to Google drive
from google.colab import drive
drive.mount('/content/drive')
```

```
# Get labeled data
import pandas as pd
import glob

data_path = "/content/drive/MyDrive/data/Doc_labeled_requirements"
data_files = glob.glob(data_path + "/*.xlsx")

for x in range(len(data_files)):
    print(data_files[x])

df = pd.concat((pd.read_excel(f,usecols="A,B") for f in data_files),
   ignore_index=True)
print(df.groupby('Label').count())
print(df)
```

```
# Get lists of requirements and their labels.
requirements = df.Requirements.tolist()
labels = df.Label.map({'Doc1':0, 'Doc2':1, 'Doc3':2, 'Doc4':3, 'Doc5':4}).
    tolist()
```

```
from sklearn.model_selection import train_test_split

# Partition into train and test sets
train_requs, test_requs, train_labels, test_labels = train_test_split(
    requirements, labels, random_state=500, test_size=.2, stratify=labels)

# Further partition train set into train and evaluation sets
train_requs, val_requs, train_labels, val_labels = train_test_split(
    train_requs, train_labels, random_state=501, test_size=.1, stratify=
    train_labels)
```

```
from transformers import BertTokenizerFast
tokenizer = BertTokenizerFast.from_pretrained('bert-base-uncased')

# Tokenize
train_encodings = tokenizer(train_requs, truncation=True, padding=True)
val_encodings = tokenizer(val_requs, truncation=True, padding=True)
test_encodings = tokenizer(test_requs, truncation=True, padding=True)
```

```
# Prepare datasets
import torch

class RequDataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in self.encodings
    .items()}
        item['labels'] = torch.tensor(self.labels[idx])
        return item

    def __len__(self):
        return len(self.labels)

train_dataset = RequDataset(train_encodings, train_labels)
val_dataset = RequDataset(val_encodings, val_labels)
test_dataset = RequDataset(test_encodings, test_labels)
```

```
# Install datasets package to load metrics
!pip install datasets
```

```
# Fine-tune
import numpy as np
from transformers import BertForSequenceClassification, Trainer,
   TrainingArguments
from datasets import load_metric

metric = load_metric("matthews_correlation")

def compute_metrics(eval_pred):
    predictions, labels = eval_pred
    predictions = np.argmax(predictions, axis=1)
    return metric.compute(predictions=predictions, references=labels)

training_args = TrainingArguments(
    output_dir='./results',          # output directory
    evaluation_strategy='epoch',      # set evaluation for each epoch
    num_train_epochs=3,              # total number of training epochs
    per_device_train_batch_size=16,  # batch size per device during
   training
    per_device_eval_batch_size=16,   # batch size for evaluation
    learning_rate=2e-5,              # learning rate for AdamW optimizer
    warmup_ratio=0.1,                # ratio of training data to use for
   warmup of learning rate scheduler
    weight_decay=0.01,               # strength of weight decay
    logging_dir='./logs',            # directory for storing logs
    logging_strategy='epoch',
)

model = BertForSequenceClassification.from_pretrained("bert-base-uncased",
    num_labels=5)

trainer = Trainer(
    model=model,                         # the instantiated Transformers
   model to be trained
    args=training_args,                  # training arguments, defined
   above
    train_dataset=train_dataset,         # training dataset
    eval_dataset=val_dataset,            # evaluation dataset
    compute_metrics=compute_metrics
)

trainer.train()
```

```
# Run model on test set
```

```
import numpy as np
predictions = trainer.predict(test_dataset)
preds = np.argmax(predictions.predictions, axis=-1)
```

```
# Compute MCC
from sklearn.metrics import matthews_corrcoef as mcc
mcc_test_labels = [label for label in test_labels]
mcc_preds = [label for label in preds.tolist()]

print(mcc_preds)
print(mcc_test_labels)
print(mcc(mcc_test_labels,mcc_preds))
```

```
# Get confusion matrix and classification report
from sklearn import metrics
print(metrics.confusion_matrix(test_labels, preds))
print(metrics.classification_report(test_labels, preds, target_names=['
    Doc1', 'Doc2', 'Doc3', 'Doc4', 'Doc5']))
```

```
# Save a trained model, configuration and tokenizer using `save_pretrained
    ()`.
import os

output_dir = './model_save/'

# Create output directory if needed
if not os.path.exists(output_dir):
    os.makedirs(output_dir)

model_to_save = model.module if hasattr(model, 'module') else model  #
    Take care of distributed/parallel training
model_to_save.save_pretrained(output_dir)
tokenizer.save_pretrained(output_dir)

!cp -r ./model_save/ "/content/drive/MyDrive/models/
    BERTdocumentclassification"
```

Appendix B

Code for Computing Similarity with PDC [CLS] Embeddings

```
# Make sure GPU is equipped
gpu_info = !nvidia-smi
gpu_info = '\n'.join(gpu_info)
if gpu_info.find('failed') >= 0:
  print('Not connected to a GPU')
else:
  print(gpu_info)
```

```
# Install transformers package in colab session
!pip install transformers
```

```
# Link colab to Google drive
from google.colab import drive
drive.mount('/content/drive')
```

```
# Get data
import pandas as pd
import glob

data_path = "/content/drive/MyDrive/data/Doc_labeled_requirements"
data_files = glob.glob(data_path + "/*.xlsx")
df = pd.concat((pd.read_excel(f,usecols="A,B") for f in data_files),
    ignore_index=True)
print(data_files)
print(df.groupby('Label').count())
requirements = df.Requirements.tolist()
```

```
# Load pretrained tokenizer and model
```

```
from transformers import BertForSequenceClassification, BertTokenizerFast
import torch

model_dir = "/content/drive/MyDrive/models/BERTdocumentclassification"

tokenizer = BertTokenizerFast.from_pretrained(model_dir)
model = BertForSequenceClassification.from_pretrained(model_dir,
    output_hidden_states=True)

if torch.cuda.is_available():
     device = torch.device("cuda")

model = model.to(device) # Copy model to GPU
```

```
# Get [CLS] embeddings
def requirement_to_embedding(model,tokenizer,requirement):
  input = tokenizer(requirement, padding=True, truncation=True,
   return_tensors="pt")
  input = input.to(device) # copy input to GPU

  output = model(**input) # run model without labels to get logits &
   encoded layers
  hidden_states = output.hidden_states
  embedding = hidden_states[12][0][0] # each layer has output of size (
   batch_zize,sequence_length,hidden_size); here we are getting the [CLS]
   token from the final layer
  embedding = embedding.detach().cpu().numpy()
  return embedding
```

```
# Make embeddings array
import numpy as np

embeddings = [requirement_to_embedding(model,tokenizer,requirement) for
    requirement in requirements]
embedd_array = np.stack(embeddings)

embedd_array.shape
```

```
# Create distance matrix between averaged doc embeddings
from sklearn.metrics.pairwise import cosine_distances

Doc1_index = df[df['Label']=='Doc1'].index.tolist()
Doc2_index = df[df['Label']=='Doc2'].index.tolist()
Doc3_index = df[df['Label']=='Doc3'].index.tolist()
Doc4_index = df[df['Label']=='Doc4'].index.tolist()
Doc5_index = df[df['Label']=='Doc5'].index.tolist()
```

```python
Doc1_embedd = embedd_array[Doc1_index,:].copy()
Doc1_mean = np.mean(Doc1_embedd,axis=0)
print(Doc1_mean.shape)

Doc2_embedd = embedd_array[Doc2_index,:].copy()
Doc2_mean = np.mean(Doc2_embedd,axis=0)

Doc3_embedd = embedd_array[Doc3_index,:].copy()
Doc3_mean = np.mean(Doc3_embedd,axis=0)

Doc4_embedd = embedd_array[Doc4_index,:].copy()
Doc4_mean = np.mean(Doc4_embedd,axis=0)

Doc5_embedd = embedd_array[Doc5_index,:].copy()
Doc5_mean = np.mean(Doc5_embedd,axis=0)

mean_embedds = np.vstack([Doc1_mean,Doc2_mean,Doc3_mean,Doc4_mean,
    Doc5_mean])
doc_similarity = cosine_distances(mean_embedds)
print(doc_similarity)
```

```python
# Install faiss
!pip install faiss
!pip install faiss-gpu
```

```python
# Prep for search
import faiss
from sklearn.preprocessing import normalize

res = faiss.StandardGpuResources() # allocate single GPU

# create array of embeddings to search
other_embedds = np.vstack([Doc1_embedd,Doc2_embedd,Doc4_embedd,Doc5_embedd
    ])
norm_embedds = normalize(other_embedds,norm='l2') # normalize embeddings
    to unit sphere

# create corresponding df of requirements
query_df = df[df['Label']=='Doc3'].copy().reset_index()
other_df = df[df['Label']!='Doc3'].copy().reset_index()

# build index
index_flat = faiss.IndexFlatIP(norm_embedds.shape[1]) # build flat CPU
    index
gpu_index_flat = faiss.index_cpu_to_gpu(res, 0, index_flat) # make it a
    GPU index
gpu_index_flat.add(norm_embedds) # add requirement embeddings
```

```
# Conduct requirement similarity search
nn = 5 # number of nearest neighbors

print(query_df.iloc[128,1])
query = normalize(Doc3_embedd[128].reshape(1,-1),norm='l2') # normalize
   query vector so that inner product returns cosine similarity
print(query_df.shape)

print(other_df.shape)
print(norm_embedds.shape)
print('')

D,I = gpu_index_flat.search(query.reshape(1,768), k=nn) # perform search

for i in range(I.shape[1]):
  result_i = I[0,i] # get row number for result
  # print(result_i)
  print('Cosine Distance: %.2f' % (1 - D[0,i]))
  print('Docmument: {}'.format(other_df.iloc[result_i,2])) # display
   parent document of result
  print('Requirement: {}'.format(other_df.iloc[result_i,1]))
  print('')
```

Appendix C

Code for Fine-tuning Bert on FC Task

```
# Make sure GPU is equipped
gpu_info = !nvidia-smi
gpu_info = '\n'.join(gpu_info)
if gpu_info.find('failed') >= 0:
  print('Not connected to a GPU')
else:
  print(gpu_info)
```

```
# Install transformers package in colab session
!pip install transformers
```

```
# Link colab to Google drive
from google.colab import drive
drive.mount('/content/drive')
```

```
# Get labeled data
import pandas as pd
import glob

data_path = "/content/drive/MyDrive/data/ForNF_labeled_requirements"
data_files = glob.glob(data_path + "/*.xlsx")
df = pd.concat((pd.read_excel(f,usecols="A,B") for f in data_files),
    ignore_index=True)
df = df.loc[(df.Label == 'functional') | (df.Label == 'nonfunctional')] #
    get rid of entries without the correct labels
print(df.groupby('Label').count())
print(df)
```

```
# Get lists of requirements and their labels.
requirements = df.Requirements.tolist()
labels = df.Label.map({'functional':1, 'nonfunctional':0}).tolist()
print(labels)
```

```
from sklearn.model_selection import train_test_split

# Partition into train and test sets
train_requs, test_requs, train_labels, test_labels = train_test_split(
   requirements, labels, random_state=500, test_size=.2)

# Further partition train set into train and evaluation sets
train_requs, val_requs, train_labels, val_labels = train_test_split(
   train_requs, train_labels, random_state=501, test_size=.1)
```

```
from transformers import BertTokenizerFast
tokenizer = BertTokenizerFast.from_pretrained('bert-base-uncased')

# Tokenize
train_encodings = tokenizer(train_requs, truncation=True, padding=True)
val_encodings = tokenizer(val_requs, truncation=True, padding=True)
test_encodings = tokenizer(test_requs, truncation=True, padding=True)
```

```
# Prepare datasets
import torch

class RequDataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in self.encodings
   .items()}
        item['labels'] = torch.tensor(self.labels[idx])
        return item

    def __len__(self):
        return len(self.labels)

train_dataset = RequDataset(train_encodings, train_labels)
val_dataset = RequDataset(val_encodings, val_labels)
test_dataset = RequDataset(test_encodings, test_labels)
```

```
# Install datasets package to load metrics
!pip install datasets
```

```
# Fine-tune
import numpy as np
from transformers import BertForSequenceClassification, Trainer,
   TrainingArguments
from datasets import load_metric

metric = load_metric("matthews_correlation")

def compute_metrics(eval_pred):
    predictions, labels = eval_pred
    predictions = np.argmax(predictions, axis=1)
    return metric.compute(predictions=predictions, references=labels)

training_args = TrainingArguments(
    output_dir='./results',           # output directory
    evaluation_strategy='epoch',       # set evaluation for each epoch
    num_train_epochs=3,               # total number of training epochs
    per_device_train_batch_size=16,   # batch size per device during
   training
    per_device_eval_batch_size=16,    # batch size for evaluation
    learning_rate=2e-5,               # learning rate for AdamW optimizer
    warmup_ratio=0.1,                 # ratio of training data to use for
   warmup of learning rate scheduler
    weight_decay=0.01,                # strength of weight decay
    logging_dir='./logs',             # directory for storing logs
    logging_strategy='epoch',
)

model = BertForSequenceClassification.from_pretrained("bert-base-uncased")

trainer = Trainer(
    model=model,                          # the instantiated Transformers
   model to be trained
    args=training_args,                   # training arguments, defined
   above
    train_dataset=train_dataset,          # training dataset
    eval_dataset=val_dataset,             # evaluation dataset
    compute_metrics=compute_metrics
)

trainer.train()
```

```
# Run model on test set
import numpy as np
predictions = trainer.predict(test_dataset)
preds = np.argmax(predictions.predictions, axis=-1)
```

```
# Compute MCC
from sklearn.metrics import matthews_corrcoef as mcc
mcc_test_labels = [label for label in test_labels]
mcc_preds = [label for label in preds.tolist()]

print(mcc_preds)
print(mcc_test_labels)
print(mcc(mcc_test_labels,mcc_preds))
```

```
# Get confusion matrix and classification report
from sklearn import metrics
print(metrics.confusion_matrix(test_labels, preds))
print(metrics.classification_report(test_labels, preds, target_names=['
    Nonfunctional', 'Functional']))
```

```
# Save a trained model, configuration and tokenizer using `save_pretrained
    ()`
import os

output_dir = './model_save/'

# Create output directory if needed
if not os.path.exists(output_dir):
    os.makedirs(output_dir)

model_to_save = model.module if hasattr(model, 'module') else model  #
    Take care of distributed/parallel training
model_to_save.save_pretrained(output_dir)
tokenizer.save_pretrained(output_dir)

!cp -r ./model_save/ "/content/drive/MyDrive/models/
    BERTfunctionalclassification"
```

Appendix D

Code for Predicting Change with FC [CLS] Embeddings

```
# Make sure GPU is equipped
gpu_info = !nvidia-smi
gpu_info = '\n'.join(gpu_info)
if gpu_info.find('failed') >= 0:
  print('Not connected to a GPU')
else:
  print(gpu_info)
```

```
# Install transformers package in colab session
!pip install transformers
```

```
# Get requirements
import pandas as pd

data_path = "/content/drive/MyDrive/data/ForNF_labeled_requirements"
data_file = data_path + "/J_Doc1_ForNF.xlsx"
df = pd.read_excel(data_file,usecols="A,B")
requirements = df.Requirements.tolist()
```

```
# Load pretrained tokenizer and model
from transformers import BertForSequenceClassification, BertTokenizerFast
import torch

model_dir = "/content/drive/MyDrive/models/BERTfunctionalclassification"

tokenizer = BertTokenizerFast.from_pretrained(model_dir)
model = BertForSequenceClassification.from_pretrained(model_dir,
    output_hidden_states=True)
```

```
if torch.cuda.is_available():
    device = torch.device("cuda")

model = model.to(device) # Copy model to GPU
```

```
# Get embeddings
def requirement_to_embedding(model,tokenizer,requirement):
  input = tokenizer(requirement, padding=True, truncation=True,
   return_tensors="pt")
  input = input.to(device) # copy input to GPU

  output = model(**input) # run model without labels to get logits &
   encoded layers
  hidden_states = output.hidden_states
  embedding = hidden_states[12][0][0] # each layer has output of size (
   batch_zize,sequence_length,hidden_size); here we are getting the [CLS]
   token from the final layer
  embedding = embedding.detach().cpu().numpy()
  return embedding
```

```
# Create embedding array
import numpy as np

embeddings = [requirement_to_embedding(model,tokenizer,requirement) for
   requirement in requirements]
embedd_array = np.stack(embeddings)

embedd_array.shape
```

```
# Install faiss
!pip install faiss
!pip install faiss-gpu
```

```
# Prep for search
import faiss
from sklearn.preprocessing import normalize

res = faiss.StandardGpuResources() # allocate single GPU

norm_embedds = normalize(embedd_array,norm='l2')

# build index with all requirements
index_flat = faiss.IndexFlatIP(norm_embedds.shape[1]) # build flat CPU
   index
gpu_index_flat = faiss.index_cpu_to_gpu(res, 0, index_flat) # make it a
   GPU index
```

```
gpu_index_flat.add(norm_embedds) # add requirement embeddings
```

```
# Search among all requirements in document
nn = df.shape[0] # number of nearest neighbors

# ECN01
requ_01a = 137
print(df.iloc[requ_01a,1])
print(df.iloc[requ_01a,0])
print('')

requ_01b = 16
print(df.iloc[requ_01b,1])
print(df.iloc[requ_01b,0])
print('')

requ_01c = 75
print(df.iloc[requ_01c,1])
print(df.iloc[requ_01c,0])
print('')

requ_01d = 19
print(df.iloc[requ_01d,1])
print(df.iloc[requ_01d,0])
print('')

# ECN07
requ_07a = 19
print(df.iloc[requ_07a,1])
print(df.iloc[requ_07a,0])
print('')

requ_07b = 24
print(df.iloc[requ_07b,1])
print(df.iloc[requ_07b,0])
print('')

# ECN11
requ_11 = 97
print(df.iloc[requ_11,1])
print(df.iloc[requ_11,0])
print('')

query_07 = norm_embedds[[requ_01a,requ_01b,requ_01c,requ_01d],:].copy()
D_07,I_07 = gpu_index_flat.search(query_07.reshape(4,768), k=nn) # perform
    search
```

```
# find ranking of 07a for each requirement in ECN01
all_ranks_07b = []
all_dists_07b = []
for i in range(I_07.shape[0]):
  ranks_07b = np.where(I_07[i] == requ_07b) # find rank of requirement in
    search array
  dists_07b = 1 - D_07[i,ranks_07b] # find distance of requirement in
    distance array
  all_ranks_07b.append(ranks_07b[0][0])
  all_dists_07b.append(dists_07b[0][0])

print(all_ranks_07b)
print(all_dists_07b)
print('')

query_11 = norm_embedds[[requ_01a,requ_01b,requ_01c,requ_01d,requ_07b],:].
    copy()
D_11,I_11 = gpu_index_flat.search(query_11.reshape(5,768), k=nn) # perform
     search

# find ranking of 11 for each requirement in ECN07
all_ranks_11 = []
all_dists_11 = []
for i in range(I_11.shape[0]):
  ranks_11 = np.where(I_11[i] == requ_11) # find rank of requirement in
    search array
  dists_11 = 1 - D_11[i,ranks_11] # find distance of requirement in
    distance array
  all_ranks_11.append(ranks_11[0][0])
  all_dists_11.append(dists_11[0][0])

print(all_ranks_11)
print(all_dists_11)
```

```
# Split FRs and NFRs prior to search

# get indeces for both classes
func_index = df[df['Label']=='functional'].index.tolist()
nonfunc_index = df[df['Label']=='nonfunctional'].index.tolist()

# sepparate functional embeddings from nonfunctional
func_embedds = norm_embedds[func_index,:].copy()
nonfunc_embedds = norm_embedds[nonfunc_index,:].copy()

# create corresponding df of requirements
func_df = df[df['Label']=='functional'].copy().reset_index()
nonfunc_df = df[df['Label']=='nonfunctional'].copy().reset_index()
```

```
# skip functional since change only propagates to NFRs here
# index_flat_func = faiss.IndexFlatIP(func_embedds.shape[1]) # build flat
   CPU index
# gpu_index_flat_func = faiss.index_cpu_to_gpu(res, 0, index_flat_func) #
   make it a GPU index
# gpu_index_flat_func.add(func_embedds) # add functional requirement
   embeddings

# D_func,I_func = gpu_index_flat_func.search(query.reshape(1,768), k=nn) #
    perform search

index_flat_nonfunc = faiss.IndexFlatIP(nonfunc_embedds.shape[1]) # build
   flat CPU index
gpu_index_flat_nonfunc = faiss.index_cpu_to_gpu(res, 0, index_flat_nonfunc
   ) # make it a GPU index
gpu_index_flat_nonfunc.add(nonfunc_embedds) # add nonfunctional
   requirement embeddings

nn = nonfunc_df.shape[0] # search all nonfunctional requirements

# find ranking of ECN07b requirement among nonfunctional requirements for
   ECN01
D_07_nonfunc,I_07_nonfunc = gpu_index_flat_nonfunc.search(query_07.reshape
   (4,768), k=nn) # perform search

nonfunc_requ_07b = nonfunc_df.index[nonfunc_df['index'] == requ_07b].
   tolist()[0] # find corresponding index in nonfunctional dataframe
print(nonfunc_df.loc[[nonfunc_requ_07b]])

nonfunc_ranks_07b = []
nonfunc_dists_07b = []
for i in range(I_07_nonfunc.shape[0]):
  ranks_07b = np.where(I_07_nonfunc[i] == nonfunc_requ_07b) # find rank of
    requirement in search array
  dists_07b = 1 - D_07_nonfunc[i,ranks_07b] # find distance of requirement
    in distance array
  nonfunc_ranks_07b.append(ranks_07b[0][0])
  nonfunc_dists_07b.append(dists_07b[0][0])

print(nonfunc_ranks_07b)
print(nonfunc_dists_07b)
print('')

# find ranking of ECN11 requirement among nonfunctional requirements for
   ECN07
```

```
D_11_nonfunc,I_11_nonfunc = gpu_index_flat_nonfunc.search(query_11.reshape
    (5,768), k=nn) # perform search

nonfunc_requ_11 = nonfunc_df.index[nonfunc_df['index'] == requ_11].tolist
    ()[0] # find corresponding index in nonfunctional dataframe
print(nonfunc_df.loc[[nonfunc_requ_11]])

nonfunc_ranks_11 = []
nonfunc_dists_11 = []
for i in range(I_11_nonfunc.shape[0]):
  ranks_11 = np.where(I_11_nonfunc[i] == nonfunc_requ_11) # find rank of
    requirement in search array
  dists_11 = 1 - D_11_nonfunc[i,ranks_11] # find distance of requirement
    in distance array
  nonfunc_ranks_11.append(ranks_11[0][0])
  nonfunc_dists_11.append(dists_11[0][0])

print(nonfunc_ranks_11)
print(nonfunc_dists_11)
```

Appendix E

Code for Obtaining SB Embeddings

```
# Install sentence transformers package
!pip install -U sentence-transformers
```

```
# Link colab to Google drive
from google.colab import drive
drive.mount('/content/drive')
```

```
# Get data
import pandas as pd
import glob

# Load requirements with functional label and append document label
data_path = "/content/drive/MyDrive/data/ForNF_labeled_requirements"
data_files_func = glob.glob(data_path + "/*.xlsx")
print(data_files_func)

df_Doc1 = pd.read_excel(data_path + "/J_Doc1_ForNF.xlsx",usecols="A,B")
df_Doc1['Document'] = 'Doc1'

df_Doc2 = pd.read_excel(data_path + "/J_Doc2_ForNF.xlsx",usecols="A,B")
df_Doc2['Document'] = 'Doc2'

df_Doc3 = pd.read_excel(data_path + "/J_Doc3_ForNF.xlsx",usecols="A,B")
df_Doc3['Document'] = 'Doc3'

df_Doc4 = pd.read_excel(data_path + "/Doc4_ForNF.xlsx",usecols="A,B")
df_Doc4['Document'] = 'Doc4'

df_Doc5 = pd.read_excel(data_path + "/Doc5_ForNF.xlsx",usecols="A,B")
df_Doc5['Document'] = 'Doc5'
```

```
df = pd.concat([df_Doc1,df_Doc2,df_Doc3,df_Doc4,df_Doc5],ignore_index=True
    )
print(df.groupby(['Label','Document']).count())
print(df)
```

```
# Create embedding array
from sentence_transformers import SentenceTransformer
import numpy as np
from sklearn.preprocessing import normalize

requirements = df.Requirements.tolist()
model = SentenceTransformer('sentence-transformers/multi-qa-distilbert-cos
    -v1')
embeddings = model.encode(requirements)
embedd_array = np.stack(embeddings)
print(embedd_array.shape)
```

Appendix F

Provided Instructions For Labeling FRs and NFRs

In the “Label” column, please enter “functional” or “nonfunctional” for each of the requirements. The requirements are sorted according to the document they came from (indicated by the “Document” column).

For guidance, use the following definition of functional requirements: “what a product must do, be able to perform, or should do.” All other requirements are considered nonfunctional here. Requirements related to the following subjects are commonly classified as nonfunctional: “reliability, security, accuracy, cultural factors, and other descriptors that do not necessarily describe actions that must be taken by the system.”

Here are some examples:

Functional
- The system shall send a verification email to a user whenever he/she registers for the first time.
- The seatbelt shall secure the passenger to their seat.
- The bicycle suspension shall absorb energy from rocky terrain.

Nonfunctional
- Emails shall be sent with a latency no greater than 12 hours.
- The seatbelt shall satisfy all local and federal regulations.
- The bicycle frame shall not fail within 10,000 hours of normal use.