# Leveraging Latent Textual Topology for Standard Identification in Engineering Design

by

Matthew Booth Bowen

(Under the Direction of Beshoy Morkos and Benjamin Wagner)

Abstract

In engineering design, standards provide essential technical definitions and guidelines, reflecting best practices recognized across the industry. These standards, developed by various Standard Developing Organizations (SDOs), encapsulate technological expertise to enhance safety, reliability, productivity, and efficiency in component and system design. However, the vast number and diversity of these documents pose challenges for designers in selecting and implementing appropriate standards due to inconsistencies, conflicting advice, and overlaps. This thesis proposes an information retrieval approach to improve product development by linking product requirements with relevant engineering standards. It explores the use of Artificial Intelligence techniques, including language embeddings, semantic search, and zero-shot learning, to efficiently navigate this extensive information corpus. Additionally, the study investigates the optimal contextual window using sentence-level and extensive project data, aiming to develop novel frameworks for more effective engagement with the textual knowledge in engineering standards, thereby aiding design engineers.

Index words: Large Language Models, Artificial Intelligence, Design Standards, Design Requirements, Design Automation, BERT, GPT

# Leveraging Latent Textual Topology for Standard Identification in Engineering Design

by

Matthew Booth Bowen

B.S.A.E., University of Georgia, 2022

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

2024

# Leveraging Latent Textual Topology for Standard Identification in Engineering Design

by

Matthew Booth Bowen

Major Professors: Beshoy Morkos
Benjamin Wagner

Committee: Haygriva Rao
Jidong Yang

Electronic Version Approved:

Ron Walcott
Dean of the Graduate School
The University of Georgia
May 2024

# Dedication

This thesis is dedicated to my parents, who instilled in me an appreciation for education and hard work.

# Acknowledgments

*"Work not for yourself, but always strive onward so that you may be able to pull another up."* - D.W. Brooks

I would like to begin by thanking all who have helped me in this endeavor. Due to the support that my friends and family have showered upon me, and the grace of God, I have been able to succeed in the completion of this work. Special thanks are due to Dr. Morkos, who was willing to take a risk on me by allowing me to start in his lab, and for his guidance and support that followed. Dr. Morkos has shown me not only how to be a good researcher but also how to be a good engineer. Dr. Wagner, thank you for being my rock and foundation throughout this work and always reminding me of what I am capable of. To Dr. Rao and Yang, I am thankful for the time that you have put into me and this work over the past months. Your insight has been invaluable. Cody Carroll and my other lab mates have been a constant source of support and welcomed me with open arms at the beginning of my Master's. Cody, thank you in particular for the countless arguments; I know they helped to keep me sharp. Thanks are certainly due to Logan, who helped me significantly in the final weeks by proofreading and helping me in the preparation of this manuscript. Last but certainly not least, I would like to thank Kendall Kay for her constant support and love throughout this challenge.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Engineers, as curators of the product design process, spend considerable time and resources navigating through the vast amounts of information stored as natural language (Jha and Mahmoud, 2019). This is particularly evident throughout the design process, where engineers make critical decisions regarding system requirements that shape the final product (Robertson and Robertson, 2012). These decisions often involve referencing engineering standards, which are integral to guiding their choices and inevitably become part of the design requirements. However, the identification of these standards is a laborious task, involving many manual information retrieval steps. For example, a common method of searching for standards that is put forward by Standard Developing Organizations (SDOs) is keyword or metadata search. This requires the engineer to know the index structure of standards well enough to know which keyword will return the most applicable response and this is where substantial valuable design time is wasted. Additionally, many standards exist that designers would not know to search for. This identified obstacle exists in several different presentations in design. Firstly, highly domain-competent engineers who are proficient in the formal design process experience the standard identification task as a time sink. Secondly, in smaller engineering firms, where the design process frequently involves less formality and technical documentation, standards are often not efficiently utilized due to a lack of awareness that applicable standards exist.

Large Language Models (LLMs) have generated substantial excitement in recent years and have increased in capability at a fast rate as the introduction of the transformer architecture (Vaswani et al., 2017) gave rise to the BERT (Devlin et al., 2018a) and GPT (Radford et al., 2018) families of models. These models host the potential to reduce time spent doing mundane tasks and are proved to outperform humans on many tasks such as code debugging (Haque and Li, 2023) and natural language content generation (Rathore, 2023). In the engineering domain, these advancements in NLP (Natural Language Processing) are readily applied in the area of engineering requirements for various useful tasks including functional and non-functional classification (Mullis et al., 2023; Shankar et al., 2010; Shankar et al., 2020), requirement reuse (Mihany et al., 2016), and requirement traceability (Tian et al., 2023). However, a review of the literature presents little exploration of the efficacy that large language models have in tasks surrounding engineering standards. Therefore, the researchers look to answer the following three research questions.

- **RQ 1** - How may requirements and standards be distinguishable through large language models?
- **RQ 2** - Do language embeddings correlations exist between engineering project requirements and standard document sub-clauses?
- **RQ 3** - Do language embedding correlations exist between conceptual engineering project descriptions with relevant standards?

Standards may arise in a design's technical documentation either implicitly or explicitly with the former not directly traceable. For example, an engineer could have inherent knowledge of the governing standards for a particular design, and this would shape how he guided the design and generated the requirements yet not explicitly reference the standard. This becomes an issue when standards, which evolve, change. Without a formal certification process, products in production may not reflect the latest design practices recommended by these standards. The first step in this traceability is automated correlation between design projects and the relevant standards. This automated, and dynamic, information correlation and retrieval task is where the researchers hypothesize that LLMs hold substantial potential. In contrast with keyword-based or metadata information retrieval, large language models offer the ability to incorporate

semantics and context into the pipeline. This is done through natural language embeddings that serve as high-dimensional representations of the meaning of a language input. The semantic search is framed as populating a continuous vector space with inputs that occupy the natural language manifold and the location of each input on the manifold is representative of the input's meaning. The interaction with this vector store is then executed by embedding the search query onto the manifold and extracting the nearest point. Theoretically, the most contextually relevant information may then be returned. The successful completion of standards retrieval, when integrated into an auto-generation process, then allows the creation of Retrieval Augmented Generation (RAG) models capable of further creating engineering documentation representing the information contained in retrieved engineering standards.

In work addressing RQ1, the BERT model is fine-tuned with a labeled dataset containing requirements and standards to evaluate the performance of this model on this task. This task, referred to as the SRC Task (Standard-Requirement Classification Task), constructs foundational knowledge of BERT's performance with this type of engineering-specific language and its potential for detecting explicit references to engineering standards for enhanced traceability. After fine-tuning the model, the researchers employ the Integrated Gradients technique to collect word attributions for the corpus. The researchers propose that this approach may help gain a more robust understanding of the black-box classification predictions yielded by the fine-tuned BERT model, in addition to evaluating Matthew's Correlation Coefficient (MCC) for the model's performance on unseen data.

RQ2 utilizes and evaluates language embeddings for the S2R Task (Standard-to-Requirement Task) within the framework of using existing project requirements as search queries. This experiment is structured to use only the prescriptive information from engineering standard documents to populate the vector space. To clarify, standard documents often contain substantial descriptive information, including scope, terminology, normative references, figures, and tables. However, the prescriptive information in these documents is usually presented as one or a few sentence statements concisely prescribing what the designer must do to satisfy the standard. These items are often referred to as standard requirements. To eliminate potential confusion in this thesis, the term 'standard requirements' will henceforth be referred to

as 'standard sub-clauses.' Notably, this approach provides the model with the minimum unit of contextual relevance for both requirements and standard documentation.

RQ 3 explores the use of language embeddings for the S2R task in the context of early design, before the generation of requirements or other prescriptive artifacts, where only high-level descriptions may exist. To satisfy this research question, three different methodologies for embedding-based search are evaluated on a ground-truth industry validation case. Additionally, by leveraging a larger context window model, and employing chunking strategies, these experiments allow for the embedding of entire long-form standard documents.

As an outcome of this research, the researchers developed the Standards Augmented Requirements Generator (SARG) model that utilizes findings from the research questions of this work. This tool uses the highest-performing embedding-based search method identified by experiments conducted to satisfy RQ 3. The SARG model further synthesizes the standard documents returned by the search method and presents the information to the designer in a more palatable, requirements-based format. Furthermore, an industry expert evaluates the SARG tool's output.

RQ 1 is evaluated on a labeled dataset generated from engineering standards and industry projects hosted by the University of Georgia Model Lab. For RQ 2, industry project requirement sets are used to evaluate language embedding-based search and are human-evaluated for relevancy. In RQ 3, an industry case study is used to evaluate the different methods. Figure 1.1 depicts the research logic map for this thesis and the tasks performed to satisfy each research question.

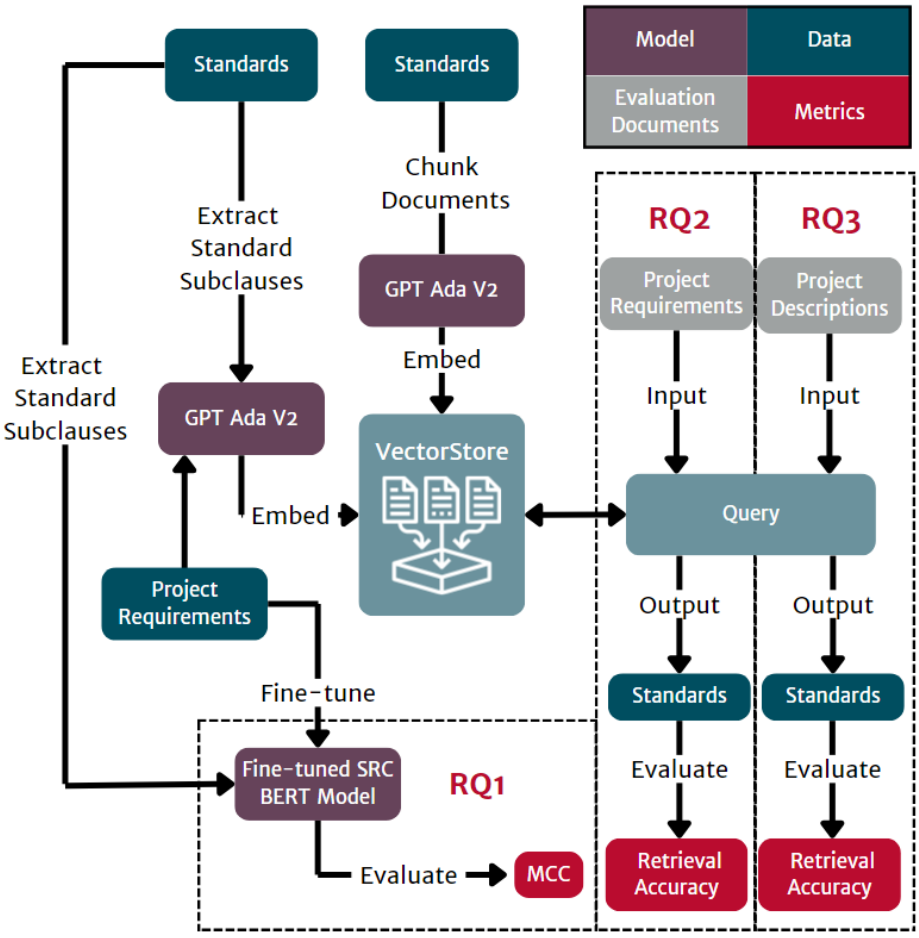


Figure 1.1: Research Logic Map

# Chapter 2

# Background

This chapter presents the foundational background of this work. Section 2.1 discusses the crucial role of standards in engineering design, detailing their development, challenges in application, the evolution of methods for managing and accessing standards, and the ongoing efforts to improve standards' interaction through technological advancements. Then, Section 2.2 discusses the critical role of engineering requirements in project design and explores the use of Natural Language Processing (NLP) methods such as BERT for enhancing requirement management. This section also highlights the importance of standards traceability and requirement reuse, underscoring the need for tools to link standards to requirements effectively, streamlining compliance and product development. Section 2.3 outlines the evolution of information retrieval from early keyword indexing to advanced deep learning methods, emphasizing the development of key concepts such as ranked retrieval, term frequency-inverse document frequency (TF-IDF), and stemming, alongside the transition to vector space models and neural networks for improved information retrieval. Section 2.4 describes the introduction of the Transformer architecture and its significance in NLP, highlighting the development of pre-trained models such as BERT and GPT for enhanced language embeddings. Lastly, Section 2.5 provides a summary of the chapter and refreshes key findings in the literature.

## 2.1 Design Standards

Standards play a crucial role in the engineering design process, encompassing technical definitions and guidelines meant for designers, manufacturers, and users, and serving as widely accepted best practices. These standards, numbering in the hundreds of thousands, represent the current state of the art and foster safety, reliability, productivity, and efficiency in both component and system design (ASME, 2024). Design standards exist in two primary presentations: formal and informal. Formal standards are derived either by consensus or from a regulatory basis, while informal standards can be framed at the enterprise level, whereby enterprise design guidelines are specified (Galley-Taylor et al., 2011).

One of the initial steps towards standardizing component design began with the creation of a system for interchangeable parts. The credit for this innovation is disputed between Eli Whitney and Simeon North, but the significance of their system is unquestionable. Their work established a method by which components were manufactured to precise standards by specialized workers and then assembled later (Coulson, 1944). This approach was originally developed to expedite the production of muskets for the nascent United States and was presented to government officials who would later mandate this uniformity across other firearm factories. The collaboration between the industry and government, followed by the adoption of this process in various facilities, marks a clear starting point for national standardization in modern American industry (Bodner and Rouse, 2009).

The early push for developing standards was significantly driven by catastrophic failures, highlighting the need for safety standards in the United States. A notable instance of this was the frequent occurrence of boiler explosions during the 19th and early 20th centuries. From 1870 to 1910, the U.S. experienced over 10,000 such explosions. In response, the American Society of Mechanical Engineers (ASME) formulated the first comprehensive boiler and pressure vessel code in 1914, which states and cities quickly adopted voluntarily (ASME, 2010). This standard significantly mitigated the safety crisis, evidenced by the fact that from 1974 to 1984, the U.S. did not report a single boiler explosion.

While the value of standards is clear, the sheer volume and diversity of these standards pose significant challenges for designers in identifying or applying the relevant standards for their projects. Additionally, these standards, developed by various Standard Developing Organizations (SDOs), can sometimes lack uniformity, offer conflicting recommendations, or even be redundant. To mitigate these issues, organizations such as the American National Standards Institute (ANSI) work to streamline the procedures of SDOs by accrediting these organizations to due process requirements for American National Standard (ANS) designation (Mcclung, 2011). This effort endeavors to standardize engineering practices and promote alignment with international applications, with oversight from entities such as the International Organization for Standardization (ISO) (Alonzo, 2010).

Furthermore, with design engineers spending over 55 percent of their time engaged with technical documentation (Xia et al., 2017), more efficient interaction methods are required to work with sources of engineering information such as engineering standards. In work done with engineers at Walt Disney Corporation, Harrs et al. documented the frustrations that engineers face when attempting to retrieve relevant standards using SDOs' existing systems and databases. In this work, a new digital library and information retrieval system was created to interact with formal and informal engineering standards and was well received by practitioners (Harrs, 2006). Notably, this referenced work illustrates well-documented accounts of a large corporation and well-experienced engineers who are unsatisfied with the current method of sourcing standards for engineering design. This demonstrates that the issue arises across the board for large enterprises and small to medium enterprises alike, with the latter often not possessing developed technical information architectures (De Jong and Marsili, 2006).

Entities hosting engineering standards have begun to recognize the user need for more efficient interaction methods and are altering their storage architectures to be machine-actionable by converting their content to eXtensible markup language (XML). An example of this effort is the Deutsches Institut für Normung (DIN), which translates in English to the German Institute for Standardization. DIN provides its standard database of 33,500 documents (DIN, 2024) hosted in XML format. Moreover, the

National Information Standards Organization (NISO) has recommended a standard XML schema, NISO Standard Tag Suite, that will guide the consistency of standards hosted in XML format (NISO, 2017).

These developments have led to a new need to evaluate how best to interact with these machine-actionable databases. Luttmer et al. evaluated various techniques for extracting prescriptive sentences from the extensive DIN standards database. The researchers evaluated supervised and unsupervised techniques on this task with the Support Vector Machine (SVM) model and fine-tuned BERT achieving the best results as measured by F1-score (Luttmer et al., 2023). While this work demonstrates promise for using BERT to mine these databases for key information, the core issue of *project-relevant standard retrieval* is not addressed. Additionally, these methods lack agility by needing to be retrained each time the database is updated. This raises serious constraints for implementation because of the evolutionary nature of standard documents and the lack of multi-database applicability of this model.

Some of the earliest work done with trying to simplify processes around engineering standards arose in civil engineering disciplines, dealing with structural codes. Early approaches included the implementation of tabular decision logic tables to help organizationally track the satisfaction of necessary structural building codes and standards (Fenves, 1966). Following this work, a prototype software was created that allowed the automated processing of design specifications in structural design. This software allowed for the management of user-generated standard libraries and integrated the tabular decision logic tables in automated flow with CAD and structural analysis (Cronembold and Law, 1988). This progression of work verifies that interest exists in developing tools to ease the strain of working with standards. However, these tools do not bridge the information retrieval gap that exists currently. Additionally, these tools require highly specialized algorithms and are best used where standards repetitively apply to many designs. More recently, a tool was prepared to assist in compiling standards documents used in the construction industry and the partial automation of regulation conformance verification (Bouzidi et al., 2012). This work presented a new domain ontology in Web Ontology Language that was used to semantically map regulatory standards to project queries and automatically evaluate compliance.This work, while closer in concept to bridging the project-relevant standard retrieval gap, still lacks practicality in implementation

outside of the defined domain due to the highly algorithmic equation compliance that is the focus of many structural domain standards.

Similarly, Großer et al. present an Object-Role Modeling approach that employs graph-based methods for the modeling of project documents as well as standard-to-requirement tailoring (Großer et al., 2022). However, this approach relies on graph-based databases, which pose significant resource issues when dealing with evolutionary documents such as requirements (B. Morkos et al., 2019; Summers et al., 2014) and standards. Furthermore, this approach has only been applied to a single use case in the space domain, where the interconnections between one project and one standard developing organization are assessed. However, most engineering projects require leveraging standards from multiple originating sources with varying ontologies, demanding a more flexible approach.

The following background sections present the fields of design requirements, information retrieval, and language representation in vector space. Section 2.2 lays the foundation for design requirements' prominence and importance in engineering technical documentation. Furthermore, requirement traceability and reuse are explored in the context of value brought by dynamically mapping standards to requirements in the standard-requirement linking task and the identification of explicit standard appearances in technical documentation. Then, Section 2.3 examines the development and current state of the art in information retrieval and, specifically, the use of text embeddings in vector space information retrieval. Section 2.4 develops the background on language representations in vector space by framing the paradigm shift brought by the introduction of Transformers and their use for creating textual embeddings. Key background is then presented for the BERT and GPT models that are at the forefront of natural language processing following the introduction of Transformer architecture. Finally, Section 2.5 provides a summary of the findings and existing gaps.

## 2.2 Design Requirements

Engineering requirements are fundamental to design and specify the purpose, goals, and constraints associated with design efforts (B. Morkos et al., 2014). These requirements serve as the formalization of

stakeholder needs and guide the product development process. Requirements serve such an important purpose in design as they are known to greatly impact a project's overall success (Beitz et al., 1996). These requirements bound the design space for potential solutions and should be testable and unambiguous (Shabi et al., 2021). The NASA Systems Engineering Handbook provides a guideline for generating requirements and describes the appropriate usage of common terminology found in requirements such as "shall", "should", or "will" (Shishko and Aster, 1995). Further details describing the formulation of requirements can be found in the "Guide for Writing System Requirements" published by the International Council on Systems Engineering (Group et al., 2019). Example product requirements specified by NASA are detailed in Table 2.1.

Table 2.1: Example Requirements (NASA, 2024)

| Example Product Requirements |
|---|
| The system shall operate at a power level of... |
| The software shall acquire data from the... |
| The structure shall withstand loads of... |
| The hardware shall have a mass of... |

Many tools for requirements management exist in various industries, aiding in the management of the requirements process from elicitation to verification (McLellan et al., 2010; B. Morkos et al., 2010a; B. Morkos et al., 2010b). Throughout project development, requirements undergo the processes of elicitation, specification, validation, and verification (Wiegers and Beatty, 2013). Requirements elicitation entails engineers working closely with stakeholders to identify explicit and implicit stakeholder and project needs. As the primary technical documentation in engineering projects, requirements are critical to project success (Beitz et al., 1996; B. Morkos and Summers, n.d.). Then, the specification of elicited requirements involves the further refinement of the requirements into clear and testable statements. Later, the validation process then initiates a review of the requirement corpus with the stakeholder for "consistency, completeness, and correctness" (Kotonya and Sommerville, 1998) and ensures mutual agreement between the engineering entity and the stakeholder. Finally, requirements verification is the process of confirming, through techniques such as inspection, modeling, or expert analysis, that the requirements are satisfied

(Terry Bahill and Henderson, 2005). Recent focus on requirement changes and their propagation has sparked significant research, leading to the development of studies exploring methods to effectively manage them Hein et al., 2021, 2022; Hein et al., 2015; Htet Hein et al., 2017; B. Morkos et al., 2012; B. Morkos and Summers, 2010; Shankar et al., 2012; Summers et al., 2014). This includes innovative work as the advent of NLP methods, such as BERT, led to the creation of many tools through research exploring these tools' interactions with information stored in engineering requirements and specification documents (Chen, 2022; Chen, Carroll, et al., 2023; Chen and Morkos, 2023; Chen et al., 2021; Chen, Wei, et al., 2023). Moreover, because of comparable functional tasks, document structure, terminology, and formality resolution, there exists significant potential for leveraging insights from the work conducted on requirements when exploring NLP tools for engineering standards. Mullis et al. demonstrated the potential of BERT in several requirement-focused tasks, including parent document classification and functional classification through fine-tuning BERT (Mullis et al., 2023). Furthermore, Mullis et al. utilized BERT textual embeddings to predict requirement change propagation throughout the hierarchy of requirements structures. While researchers have made significant advances in understanding the application of large language models to engineering requirements, a gap remains in comprehending how LLMs could be utilized more efficiently for the retrieval and synthesis of engineering standards. With the foundation developed for engineering requirements, the following section will discuss the implications of the traceability of standards and their utility in facilitating requirement reuse.

### 2.2.1 Standards Traceability and Requirement Reuse

Variant products in a domain may often be subject to adherence to the same set of standards governing that field. Furthermore, standards often exist at a higher level of abstraction than product requirements in that multiple requirements must be satisfied to satisfy a standard. Figure 2.1 demonstrates different network connection components that may exist in a project. To further clarify these network components, explicit examples are provided below. In the first case, whereby the network connection is singular, REQ 1 is derived from and wholly satisfies STD 1.

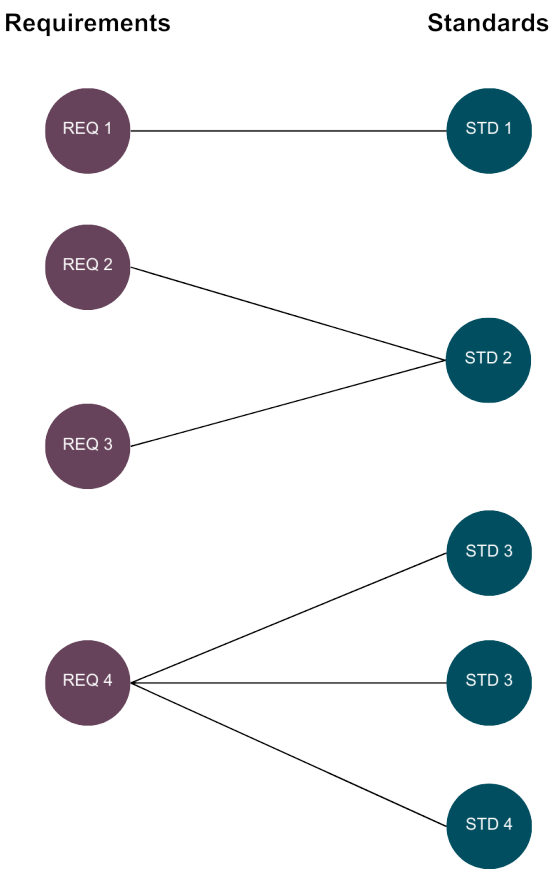


Figure 2.1: Requirement-to-Standard Network Components

- **REQ 1** : The system shall be equipped with a manually operable shutoff valve located between the mechanical air vent and the pipeline.
- **STD 1** : Where it would not be possible or practical to drain a pipeline to replace or repair mechanical air vents, a shutoff valve should be installed between the pipe and the valve.

In the second case, whereby the satisfaction of one standard is dependent upon the satisfaction of multiple requirements, an example is provided below. Similar logic may be applied in the third case.

- **REQ 2** : The container shall prominently display the wire manufacturer's name and/or brand, distinct from other content.

- **REQ 3** : The container shall include concise instructions for storage and handling, placed separately from the manufacturer's label.

- **STD 2** : The container shall also carry, in a different location, the name and/or brand name of the wire manufacturer, and necessary instructions for storing and handling.

While research surrounding the creation of tools for standard interaction remains in the nascent stages, researchers have begun to recognize the need for more advanced tools and techniques. Rouland et al highlighted current issues stemming from a lack of standard-to-requirement traceability in software design. In this work, an iterative and evolutionary process was proposed to aid the elicitation and management of requirements based on standards in system software architecture. Moreover, the researchers highlight the need for the development of tools to aid in standard-to-requirement traceability (Rouland et al., 2023). In the medical device domain, Hauksdóttir et al, reaffirmed this need for traceability from product requirements to standards and highlighted that current requirement traceability frameworks do not incorporate this need (Hauksdóttir et al., 2016). The primary contribution of this work was the development of a reusable requirements (derived from standards) catalog schema to support practitioners in reuse. However, this tool requires the manual selection of initial relevant standards and the creation of generalized requirements extracted from standard documentation. Furthermore, the researchers acknowledge that their approach does not provide adequate traceability and that a more advanced approach is needed. With the clear necessity for advanced standard interaction tools established, Section 2.3 details the chronological advancement in the information retrieval domain, highlighting the revolutionary developments in vector space information retrieval, which is utilized in the standard-requirement linking task in this work. Then, Section 2.4 explores the development of the Transformer architecture for creating dense textual representations, in the context of utilizing these embeddings for vector space information retrieval operations. Section 2.4 also accentuates the importance of Transformers in the evolution of paradigmatic pre-trained large language models (PLMs), providing a foundation for two existing PLMs utilized in this work, namely BERT and GPT.

## 2.3 Information Retrieval

Information retrieval may be succinctly defined as the process of locating information pertinent to a user's query (Sanderson and Croft, 2012). This field has seen significant evolution driven by the fundamental need to efficiently manage and retrieve information from expanding digital document collections. Furthermore, the fundamental task existing in the field of information retrieval is the ad-hoc retrieval task in which a user's query is used to search a collection of documents for relevance (Guo et al., 2016).

Early work in IR (Information Retrieval) arose in the domain of library management. An early innovation that had radical effects was libraries' shift to using keywords, derived from subject headings for document indexing, as opposed to the current system of alphabetical arrangement (Taube et al., 1952). However, the Boolean retrieval system, initially used for querying indexed documents, soon revealed its limitations in terms of flexibility and user-friendliness. In response to the poor capability of Boolean retrieval, researchers proposed an alternative approach with the concept of ranked retrieval. In ranked retrieval's nascent stages, as proposed by Luhn et al, ranked retrieval returned documents that were top-ranked due to their relevancy to the user query. This approach employed statistical methods and required manual labeling of keywords in the document collection, in addition to the assignment of weight based on the importance of the keyword to the document (Luhn, 1957). The effectiveness of this ranked retrieval method, demonstrated by Maron et al. through testing on 39 unique queries, demonstrated its superiority over Boolean retrieval. (Maron et al., 1959). Another significant contribution by Luhn et al during this period was the concept of word occurrence frequency in a document furnishing an important metric of word significance (Luhn, 1958). This would later become known as term frequency (TF) weighting and hold significance in future IR research efforts and applications. This concept laid the groundwork for future research in IR, leading to the development of inverse document frequency (IDF).

The development of inverse document frequency (IDF) was based on the concept that the frequency of word occurrence across a collection of documents was inversely proportional to its significance for retrieval (Sparck Jones, 1972). In essence, this indicated that less common or more specific words present

within the document collection were more likely to yield valid responses to user queries. Researchers found that amalgamating these metrics into Term Frequency-Inverse Document Frequency (TF-IDF), where the resulting score reflects the importance of a term for a document in the collection, to be superior to their individual use (Salton and Yang, 1973). This development represented a breakthrough in IR, as this allowed for a more nuanced understanding of term importance within and across documents. However, an early limitation of this method was that different linguistic forms of a word could dilute the significance of the term's core concept, leading to less effective retrieval outcomes. In light of this limitation, IR researchers began to explore the utility of stemming to overcome the limitations brought by word form variations.

Stemming algorithms, such as the Porter Stemmer, were developed to consolidate different morphological variants of a word to their base or root form (Porter, 1980). This approach significantly enhanced the system's ability to match lexical variants of words, thereby improving the precision of IR systems. By addressing the variability in word forms, stemming provided a critical complement to the TF-IDF model, ensuring that documents containing different forms of relevant terms could be more accurately identified and ranked in response to a query. This rule-based approach to word matching became pivotal in refining the effectiveness of IR systems, demonstrating the ongoing evolution of strategies to optimize document retrieval in the face of linguistic complexity.

Until the mid-2000s, statistical language models predicated on N-gram methodologies constituted the predominant framework for tasks within natural language processing (NLP). N-gram methods estimate the probability of a sequence of words by calculating the conditional probability of each subsequent word based on its antecedent N-1 words. Specifically, bigrams (N=2) incorporate the immediately preceding word, whereas trigrams (N=3) incorporate the two preceding words for probability estimation. This conditional probability is derived by enumerating the occurrences of N-length sequences within a corpus, subsequently normalizing these counts by the frequency of sequences initiating with identical N-1 words. Although an increment in N typically correlates with enhanced model performance, the computational complexity associated with higher N values frequently constrains N to a maximum of 5. The relative

methodological simplicity of N-gram approaches enabled the training of models on voluminous corpora, a notable example is Google's 2006 release of a 5-gram dataset derived from a corpus encompassing over 1 trillion words (Brants and Franz, 2006). Despite the scale of such models, N-grams exhibit limitations in adapting to the linguistic evolution that characterizes natural language, manifested by the generation of novel word sequences. Hence, N-grams often exhibit limitations in transferring learned patterns from a training dataset to a test set that contains novel sequences, resulting in an insufficient representation of the complex meanings inherent in language.

The following section in this chapter will address advancements made in IR following the introduction and adaptation of early vector space information retrieval techniques. This section details key foundational developments that paved the way for the development of the revolutionary Transformer architecture, which is further detailed in Section 2.4.

### 2.3.1 Vector Space Information Retrieval

Before the advent and adoption of vector space models and the later integration of deep learning methods into IR, there were significant limitations in the state of the art in handling the complexity and variability of language. Existing models and techniques struggled with linguistic variations such as synonyms, polysemy, and different word forms often leading to imprecise search results. Additionally, these models could not account for the context in which words were used, making discerning the relevance of documents beyond keyword frequency challenging. The limitations of these early approaches brought to light the need for more advanced models that could more effectively process the nuances of human language.

Advancements in the IR domain achieved substantial progress with the introduction of vector space models, notably through the implementation of the bag-of-words (BoW) approach. By representing documents and queries as sparse term vectors, where each unique word in the corpus contributes to the vectors' dimensionality, BoW facilitated a substantial methodological shift (Salton, 1962). This model abstracted document content into a mathematical form, allowing for the efficient processing and comparison of documents on a scale previously unattainable with traditional methods. Although BoW marked

a significant departure by emphasizing mathematical abstraction over linguistic precision, BoW did not fully address the need for contextual awareness in language processing (Salton et al., 1975). Nonetheless, the adoption of BoW and the optimization of sparse term vectors laid the groundwork for further innovations in IR, initiating the onset of a major paradigm shift towards incorporating neural network techniques to effectively capture the context and semantics of language.

In this progression toward more sophisticated information retrieval (IR) methodologies, the advent of word2vec marked a pivotal advancement. Developed by Mikolov et al., word2vec introduced two innovative architectures: Continuous Bag of Words (CBOW) and continuous skip-gram. These architectures significantly deviate from prior models by representing language within a continuous vector space, rather than the discrete spaces characteristic of N-gram models. This representation is crucial for capturing the nuanced relationships between words, addressing the earlier challenges of linguistic variation and the contextual relevance of terms that had hindered traditional IR models. The CBOW model, for instance, predicts a target word based on its surrounding context, both preceding and succeeding words, without regard to their sequence. Conversely, the skip-gram model reverses this approach, predicting the surrounding context provided a target word, thereby generating multiple training instances from a single input word. This method enhances the model's ability to understand the broader context in which words appear, which served as a critical advancement over the bag-of-words model's limitations in contextual processing. While the development of word2vec represented substantial progress in the field, the model held limitations, primarily because the model architecturally ignored the morphology of words.

Researchers attempted to address the limitations of word2vec with the development of Global Vectors for Word Representation (GLoVe). GLoVe architecture relies on a global matrix factorization and local context window methods by the use of global word-word co-occurrence statistics derived from the corpus (Pennington et al., 2014). In the GLoVe model's optimization process, a loss function, which is a mathematical method evaluating the model's accuracy by comparing predicted outcomes to actual ones, includes a weighting mechanism. This mechanism moderates the model's focus on both infrequent and highly frequent word co-occurrences. This training method allowed the model to have higher efficacy in

representing words that can exist in different contextual frames. However, the vector representations from GLoVe are still static embeddings, in that each word is assigned a single vector regardless of its context. For example, the word "bank" would have the same representation whether used in a financial context "bank account" or in a geographical context "river bank". This demonstrates that GLoVe is still limited in grasping the nuances of language derived from contextual usage.

The limitations discussed in this section were primarily addressed by the introduction of the Transformer architecture and the development of multi-head self-attention mechanisms (Vaswani et al., 2017). The following section will explore the Transformer architecture and the development of pre-trained LLMs with the utilization of deep learning. Then, in Subsection 2.4.1 and Subsection 2.4.2 two primary pre-trained LLMs that are utilized in this work, BERT, and GPT, are detailed.

## 2.4 Language Representation In Vector Space

The introduction of the Transformer architecture by Vaswani et al signifies paradigmatic transformation in the representation of natural language and contextual meaning (Vaswani et al., 2017). This Transformer architecture has driven innovation in the NLP field because of the superiority of multi-head self-attention mechanisms. This mechanism processes an input sequence by applying multiple attention heads that independently compute attention scores, which allows semantic features to be captured. The generated attention scores determine how the model should weigh attention to other parts of the sequence when processing a specific element. Furthermore, by utilizing multiple heads, this mechanism can attend to different positions of the input sequence. The outputs of all heads are then concatenated and linearly transformed to produce the final output allowing the model to integrate information from different representational spaces. Figure 2.2 presents a visual representation of multi-head self-attention mechanisms. By employing self-attention mechanisms, Transformers can process all parts of the input sequence simultaneously, thus making Transformers vastly more efficient for large-scale language tasks. The ability to apply parallelization allows for faster training and the handling of longer inputs without the constraints brought by the sequential nature of earlier RNN and LSTM-based methods. Because of the parallel processing

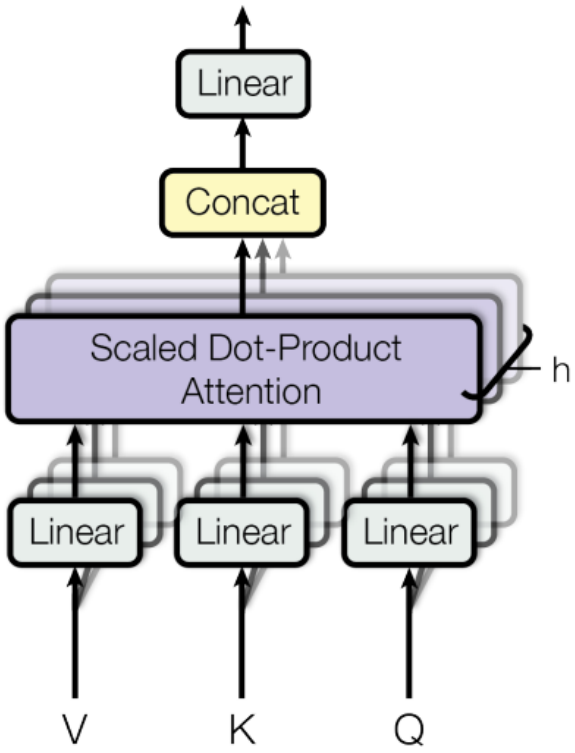


Figure 2.2: Multi-head Self-attention Mechanisms (Vaswani et al., 2017)

capabilities of Transformers and the drastically reduced resources needed to train a model, training models on previously unfeasible large text corpora became possible. This effectiveness in handling large datasets gave rise to pre-trained LLMs that are trained on general tasks and then leveraged on specific applications via fine-tuning. This concept of pre-training on generalized language tasks and then fine-tuning was a foundational development in the shift toward LLMs and is often referred to as Transfer Learning (Pan and Yang, 2009). Figure 2.3 presents a depiction of the Transformer architecture whereby the encoder and decoder portions of the model are represented.

The following sections explore two prominent PLMs, BERT and GPT, along with Retrieval Augmented Generation (RAG) in greater detail followed by Section 2.5 summarizing the findings of this chapter.

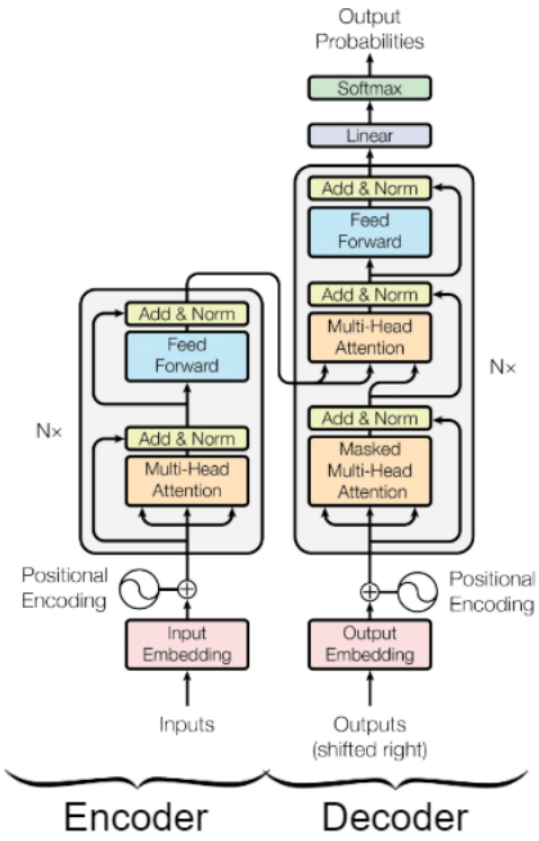


Figure 2.3: Transformer Encoder-Decoder Architecture (Vaswani et al., 2017)

### 2.4.1 Bidirectional Encoder Representations from Transformers (BERT)

The BERT model was introduced in 2017 by Devlin et al and sought to create a more effective language model with an innovative bidirectional approach in training (Devlin et al., 2018b). The bidirectional methodology of BERT allows the model to learn the context of unlabeled text from both the forward and backward positions in a sequence. Bert is an encoder-based model trained with the objective of predicting a masked word based only on the context of the nearby tokens. The encoder receives each of the masked tokens that are conditioned on the remaining tokens in the sequence. This process makes the decoder (Figure 2.3) unnecessary, as each masked token is analyzed in the context of the other tokens in the sentence. BERT was trained with two separate tasks during pre-training: the "masked language model" task and "next sentence prediction" task over a corpus of 3,300M words. Figure 2.4 depicts the pre-

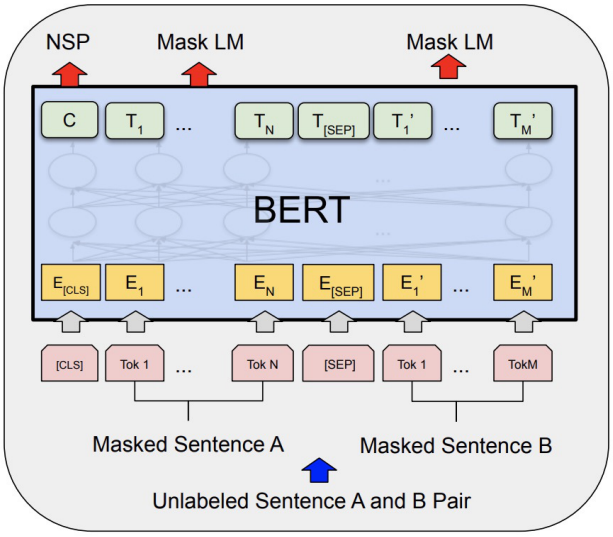


Figure 2.4: BERT Pre-training (Vaswani et al., 2017)

training process used for the unsupervised training of BERT, specifically the next sentence prediction task. The next sentence prediction task involves inputting two sentences, separated by a special "[SEP]" token, 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. The masked language model task involves randomly replacing a word in the input sequence with a special "[MASK]" token and then training the model to predict the replaced word. This task is what allows BERT to create word embeddings based on the bidirectional context of the forward and backward word in the sequence. The BERT model family contains multiple variants, including ALBERT (Lan et al., 2019), RoBERTa (Liu et al., 2019), ELECTRA (Clark et al., 2020), DistilBERT (Sanh et al., 2019), SpanBERT (Joshi et al., 2020), TinyBERT (Jiao et al., 2019), and Sentence-BERT (sBERT) (Reimers and Gurevych, 2019).sBERT represents an advanced modification leveraging a siamese structure, which processes two or more texts in parallel, within a BERT network to derive semantically significant sentence embeddings. This approach is especially effective for semantic search applications (Reimers and Gurevych, 2019). Furthermore, sBERT-based semantic search successfully supports IR tasks including coronavirus information retrieval (Esteva et al., 2021), IR for infrastructure damage queries (Kim et al., 2022), and the retrieval of relevant context from NASA's Lessons Learned Information System (Walsh and Andrade, 2022).

### 2.4.2 Generative Pre-Trained Transformer (GPT)

Brought to recent popularity by the rise of OpenAI's ChatGPT, the GPT family of models is another state-of-the-art method for NLP. While BERT is comprised of a stack of encoder blockers, GPT models are constructed with decoder blocks. In these GPT models, the multi-head attention module with cross attention is removed from the decoder due to the lack of an encoder in the pipeline. Because of this architecture, the masked multi-head self-attention is only applied to the previous words in the input sequence. The family of GPT models was initiated by the release of GPT-1 in 2018, followed by GPT-2 in 2019, GPT-3 in 2020, GPT-3.5 in 2022, and GPT-4 in 2023. Furthermore, OpenAI introduced ADA-002 in 2022, its second-generation model for embeddings-as-a-service, focusing on text embeddings. This model is specifically recommended for tasks involving text similarity by OpenAI (2022). ADA-002 has demonstrated superior performance in comparison to its predecessor, the first-generation embedding model text-similarity-davinci-001, in most text similarity tasks. ADA-002 is also recognized for its greater cost efficiency and reduced need for computational resources (Greene et al., 2022). OpenAI's initial embedding models were based on GPT technology (Neelakantan et al., 2022). Despite ADA-002's status as a commercial product of a proprietary nature, with no public disclosure of its architecture, training data, or specific details following its late 2022 launch, its relevance has increased alongside the growing interest in Large Language Models (LLMs) and embedding services. The research performed in this thesis incorporates ADA-002 for semantic similarity analysis, noting that its embeddings feature 1536 dimensions (Greene et al., 2022), which the researchers must assume were derived from an LLM.

### 2.4.3 Retrieval-Augmented Generation (RAG)

Recent advances in LLMs have led to the prevalence of models capable of generation, a capability exemplified by the GPT series. In recent years, researchers have suggested facilitating the models with the capability of accessing internal memory via some IR techniques, so that they may acquire grounded information in the generation process (Gu et al., 2018). This concept is now commonly known as Retrieval-Augmented

Generation (RAG). RAG has achieved state-of-the-art performance in many NLP tasks and attracted the attention of the scientific community (Cai et al., 2021). Compared with traditional generation models, this type of model has proven to hold important advantages (Li et al., 2022). Firstly, knowledge need not be implicitly stored in the model parameters, but can instead be acquired and leveraged upon request, leading to a superior potential in scalability. Second, RAG models generate text from a retrieved human-written reference instead of from scratch, potentially allowing more complex and contextually relevant text generation.

## 2.5 Summary

Existing studies demonstrate the need for improved tools for the retrieval and leveraging of engineering standards in design project management. Through a thorough review of the information retrieval domain, researchers identify semantic search via sequence embeddings as a potential technique for the Standard-Requirement Linking task. In this context, researchers identified OpenAI's ADA-002 language embeddings for evaluation in linking requirements with both structured formal requirements and standard searches with product conceptual descriptions. BERT has proven useful in many tasks within requirements engineering, leading to enhanced processes. However, researchers have not yet evaluated these NLP methods for tasks related to engineering standards. Therefore, BERT embeddings will be assessed for their ability to dynamically distinguish between requirements and standards. Furthermore, this work endeavors to determine if sufficient semantic context exists in standard sub-clauses and individual requirements to accurately link user-generated requirements to standards for dynamic validation of standard compliance and, inversely, for the reuse of requirements via standards. Additionally, to assess the efficacy of a tool designed to bridge the gap in project-relevant standard retrieval during the early stages of design, where technical documentation such as requirements may not yet exist, a semantic search framework utilizing OpenAI's ADA-002 embeddings will be evaluated. This evaluation will focus on conceptual project descriptions and the relevancy of the returned standards to the user's query.

# Chapter 3

# Methods

This chapter details the methodology underpinning the research conducted for this thesis. For reference, readers may refer to 1.1 for an illustration of the models, tasks, metrics, and data used in the experiments of this research. The chapter begins with a review of the completed experiments' various data in Section 3.1. Then, Section 3.2 details the methodology surrounding the fine-tuning of BERT for the SRC task. Section 3.3 encompasses various techniques utilized for evaluating the fine-tuned BERT model. The methodology for the S2R task is described in Section 3.4, detailing both the execution of the experiment and the evaluation of the approach. The execution and evaluation of three embedding-based search approaches are then described in Section 3.5. Finally, Section 3.6 presents the model architecture and evaluation for the SARG tool, an outcome of this research.

## 3.1 Review of Standard and Requirement Corpus

This section presents detailed descriptions of the requirements and standards datasets utilized in each evaluated task of this research. The researcher's intent was to assemble datasets that are diverse and extensive within feasible boundaries. Recognizing the significant impact of dataset composition on the results of this study, comprehensive information is provided for each dataset.

### 3.1.1 Labeled Dataset for Fine-tuning BERT

This section outlines the training corpus used to fine-tune the BERT model for the Standard-Requirement Classification (SRC) Task. Table 3.1 presents relevant statistics for the requirements included in the training corpus. Projects 1-4, originating from private industry, are confidential. However, these requirements have been cited in numerous other publications (Beshoy Morkos and Summers, 2012; Chen and Morkos, 2023; Hein et al., 2018; Htet Hein et al., 2017; B. Morkos et al., 2014; B. W. Morkos, 2012; Mullis et al., 2023). Projects 5 and 6 consist of publicly available requirements for design subsystems within the Square Kilometre Array (SKA) project which is set to be the world's largest radio telescope. Project 1 compiles design requirements for a threaded pipe manufacturing production line and is the most extensive, having 250 requirements. Project 2 focuses on design requirements for manufacturing stations and equipment for exhaust gas flaps. Project 3 enumerates the design requirements for industrial textile manufacturing equipment and while being fourth in the order of magnitude of requirements contains the second-highest vocabulary size, indicating a diversity in words and terminology used. Project 4 describes the design requirements for a material handling system in the pipe threading manufacturing process and is the least extensive with 37 requirements. On average, Project 6 contains the longest requirements, with an average token length of 66.33, while Project 4 contains the shortest, with only an average token length of 21.03, though this may be influenced by the substantially fewer requirements present in that document.

Table 3.1: Classification Task Requirement Training Corpus

| ID | Quantity | Average Token Length | Vocabulary Size |
|---|---|---|---|
| Project 1 | 350 | 22.65 | 1109 |
| Project 2 | 159 | 35.39 | 1132 |
| Project 3 | 214 | 31.41 | 1517 |
| Project 4 | 37 | 21.03 | 318 |
| Project 5 | 289 | 39.85 | 1477 |
| Project 6 | 291 | 66.33 | 1661 |

Table 3.2 details the standard sub-clauses used in the training corpus to fine-tune the BERT model for the standard-requirement classification task. This table provides a breakdown of the datasets utilized to fine-tune the BERT model. Each entry in the table represents a different standard or set of standards,

indicated by an acronym. The 'Quantity' column lists the number of sub-clauses or documents in the training corpus for each standard. The 'Average Token Length' column indicates the average length of the tokenized text, which may reflect the complexity or verbosity of the language in each standard. Finally, the 'Vocabulary Size' column provides insight into the lexical diversity of each standard's corpus, which may have implications for the model's ability to learn and generalize from the data. Overall, this table summarizes the composition of the training data that will inform the model's understanding of various standards in the standard-requirement classification task.

Table 3.2: Classification Task Standard Training Corpus

| ID | Quantity | Average Token Length | Vocabulary Size |
|---|---|---|---|
| AASHTO | 133 | 62.44 | 1654 |
| AATCC | 37 | 60.70 | 669 |
| AHRI | 20 | 68.35 | 408 |
| ASABE | 131 | 54.60 | 1300 |
| ASTM | 231 | 59.40 | 1914 |
| AWWA | 238 | 66.70 | 2089 |
| IEEE | 107 | 81.36 | 1559 |
| ISO | 17 | 60.65 | 283 |
| MilSpec | 9 | 52.67 | 185 |
| CCSDS | 46 | 64.49 | 548 |

Table 3.3 summarizes key metrics for the compiled standard-requirement dataset used for training the BERT model. The 'Token Length' statistics provide insights into the distribution of the text length across the dataset after tokenization, which is a crucial pre-processing step for BERT's input preparation. The mean token length of 48.89 suggests that most of the text data is concise, whereas the maximum token length of 414 indicates the presence of some more lengthy text items. The reader is reminded that the items in this dataset are divided into their minimum contextual unit. For example, a common occurrence in standard documents is where a standard sub-clause contains a level of prescriptive hierarchy lower. The comparatively long max token length of the dataset may be attributed to this occurrence, though rare. A median token length of 37.0 indicates that half of the texts are below this length, which is expected given the usual brevity of standard-requirement texts. The 'Vocabulary Size' metric measures the diversity

of the dataset's language, with a mean size of 34.525 indicating a modestly varied lexicon used across the dataset. The maximum vocabulary size of 186 reflects the richest document in lexical diversity.

Table 3.3: Summary of the Standard-Requirement Corpus Metrics

| Statistic | Token Length | Vocabulary Size |
|---|---|---|
| Mean | 48.89 | 34.525 |
| Median | 37.0 | 30.0 |
| Max | 414 | 186 |

### 3.1.2 Standard-Requirement Linking Corpus

For the standard-requirement linking experiment, the researchers populated the vector store with the standard subclauses also utilized in the SRC task. Key statistics for this dataset are described in 3.2. While this dataset contains substantial diversity and content, the researchers acknowledge that this is not a perfect representation. However, the researchers believe that the dataset is comprehensive enough to provide insight into language embedding efficacy for linking standards and requirements. Key limitations in creating a more expansive corpus for this task include the following: standards often exist behind paywalls, limiting researchers to only procuring standards licensed through the University of Georgia, and the intensive manual labor involved in extracting these sub-clauses from standard documents. Various automated approaches were evaluated for this extraction; however, the heterogeneity of document structure and hierarchical order led to little uniformity in outcome across the different document structures. With requirements serving as the input for this evaluation, researchers randomly selected five requirements from each industry project detailed in Table 3.1. This resulted in a combined 30 requirements that represent the diversity of all projects while allowing for reasonable evaluation by human annotators.

### 3.1.3 Standard Vector Store for Search Evaluation

For the evaluation of standards search methods, the researchers compiled an expansive corpus of standard documents spanning various engineering domains. This corpus has been systematically curated to

facilitate the assessment of standard search methodologies. The contents and structure of this corpus are delineated in Table 3.4. The table enumerates each Standard Developing Organization (SDO), the quantity of documents sourced, and the number of chunks into which the documents for each SDO have been segmented.

The 'chunks' mentioned in the table are defined as fixed segments consisting of 1000 tokens each. Such standardized segmentation allows for consistent contextual analysis across different documents, enabling more equitable comparisons of search methods. Each chunk acts as a window of text within a document, serving as an individual unit for retrieval and analysis in the testing of search algorithms. While the dataset is not claimed to be exhaustive or without limitations, it showcases a considerable range of variability in document length and complexity. This diversity is crucial for the thorough evaluation of search tools, ensuring they are challenged with a broad spectrum of documents to reflect a multitude of real-world conditions.

Table 3.4: Standard Vector Store Contents

| ID | Document Quantity | Chunk Quantity |
|---|---|---|
| AASHTO | 497 | 62.44 |
| AATCC | 175 | 60.70 |
| AHRI | 20 | 68.35 |
| ASABE | 10 | 89 |
| ASTM | 354 | 59.40 |
| AWWA | 173 | 66.70 |
| CCSDS | 26 | 81.36 |
| IEEE | 22 | 60.65 |

## 3.2 Fine-tuning BERT for Standard-Requirement Classification

The fine-tuning of large pre-trained models often allows better results in specific data sets (Dunn et al., 2022). The initial standard and requirement populated dataset was divided into training and testing datasets with 80% and 20% respectively allocated. Then, the training dataset was further partitioned into training and validation datasets, in which 90% percent and 10% percent were respectively allocated. The

items contained in the datasets, requirements and standard document sub-clauses, were not preprocessed beyond tokenization. Tokenization splits words in the dataset into "subtokens" that comprise BERT's pre-trained vocabulary. These subtokens are typically pieces of words that are the basic units of language that BERT uses for embeddings.

This task uses the Python interface for BERT for Sequence Classification by Hugging Face (Wolf et al., 2020), which consists of 12 encoded layers and 12 self-attention heads. The labeled dataset as described in Table 3.1 and Table 3.2 was divided into test, train, and validation portions using the scikit-learn train_test_split function (Pedregosa et al., 2011). The dataset was transformed from a labeled data frame into a comma-separated values (CSV) format. This transformation facilitated processing with the BertTokenizer class's from_pretrained method, provided by Hugging Face, utilizing the bert-base-uncased pre-trained model (Wolf et al., 2020). The dataset was labeled with binary values, '0' for 'standard' and '1' for 'requirement', and then used to train the model. The training process was regulated by an early stopping mechanism, which terminated training if the improvement in the Matthews Correlation Coefficient (MCC) was less than 0.01, to prevent overfitting and ensure training efficiency. The AdamW optimizer was employed to minimize the model's loss function during training. Unlike the traditional Adam optimizer, AdamW introduces a decoupling of weight decay from the optimization steps, which can lead to better training stability and model performance, especially in complex models and datasets (Loshchilov and Hutter, 2017). The specific parameters used for training are detailed in Table 3.5.

Table 3.5: Standard Requirement Classification Fine-tuning Parameters

| Parameter | Selection |
|---|---|
| Warmup Ratio | 0.1 |
| Weight Decay | 0.01 |
| Optimizer | AdamW |
| Learning Rate | 2e-5 |
| Metric | MCC |

## 3.3 Analysing Fine-tuned Model

Given the datasets' mild imbalance in token length and quantity, the Matthews Correlation Coefficient (MCC) was used to evaluate the model's performance accurately. Detailed in Equation (3.1), MCC includes true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) in its calculation, providing a balanced performance measure that is particularly valuable for unbalanced datasets.

$$\text{MCC} = \frac{(TP \times TN) - (FP \times FN)}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{3.1}$$

The MCC value ranges from -1 to 1, with 1 indicating perfect classification accuracy, 0 equivalent to random chance, and -1 indicating complete disagreement between predictions and actual outcomes.

### 3.3.1 Calculation of Word Attributions by Integrated Gradients

In addition to assessing MCC to evaluate the performance of the fine-tuned model, the research team utilized the Integrated Gradients technique, a method rooted in neural network interpretability, to gain deeper insights into how the model trained on the labeled dataset. This approach involves analyzing word attributions, which sheds light on the influence of specific words on the model's predictive accuracy across different classes (standard or requirement). By identifying which words significantly affect the model's ability to accurately predict the correct class, researchers can better understand the underlying mechanisms of the model's decision-making process. Integrated Gradients attribute the prediction of a neural network to its input features by integrating the gradients for the inputs along the path from a baseline $x'$ to the actual input $x$. The attribution of an input feature $x_i$ is computed as follows:

$$\text{Attribution}(x_i) = (x_i - x_i') \times \int_{\alpha=0}^{1} \frac{\partial F(x' + \alpha \times (x - x'))}{\partial x_i} d\alpha$$

where: - $x_i$ is the $i$-th feature of the input $x$, - $x_i'$ is the $i$-th feature of the baseline input $x'$, - $F$ is the model function, - $\frac{\partial F(x)}{\partial x_i}$ is the gradient of $F$ with respect to $x_i$, - $\alpha$ scales the input from the baseline $x'$ to

the actual input $x$, - The integral accumulates the gradients at all points along the path from the baseline to the input.

In practice, the integral is approximated using a discrete summation over $m$ steps:

$$\text{Attribution}(x_i) \approx (x_i - x_i') \times \sum_{k=1}^{m} \frac{\partial F(x' + \frac{k}{m} \times (x - x'))}{\partial x_i} \times \frac{1}{m}$$

where $m$ is the number of steps used in the approximation, and the path from $x'$ to $x$ is discretized into $m$ small steps.

$$\bar{A}_{w,c} = \frac{1}{N} \sum_{i=1}^{N} A_{w,c}^{(i)}$$

where: - $\bar{A}_{w,c}$ is the average attribution of word $w$ for class $c$, - $A_{w,c}^{(i)}$ is the attribution of word $w$ for class $c$ in the $i$-th text, - $N$ is the total number of texts where word $w$ appears and class $c$ is considered.

If considering two classes, $c_1$ and $c_0$ (for instance, "standard" and "requirement"), the difference in attributions for word $w$ is:

$$D_w = \bar{A}_{w,c_1} - \bar{A}_{w,c_0}$$

where: - $D_w$ is the difference in average attributions between classes $c_1$ and $c_0$ for word $w$, - $\bar{A}_{w,c_1}$ is the average attribution of word $w$ for class $c_1$, - $\bar{A}_{w,c_0}$ is the average attribution of word $w$ for class $c_0$.

Words present in the corpus may be then ranked by the absolute value of their difference in attributions to highlight those with the greatest impact on class distinction.

### Visualizing Word Attributions

After the extraction of the word attributions from the fine-tuned model, the researchers utilized a histogram and word clouds to visualize the results. The histogram displays the frequency of word attribution weights and provides insight into the distribution of the data and was plotted using Python's Seaborn visualization library. The attribution weights were filtered for outliers before the creation of the word

cloud visualizations as shown by Figure 4.5 and Figure 4.6. To filter the data, the Interquartile Range (IQR) method was utilized to remove the outliers. The filtered attributions were then separated into positive and negative attributions based on their values. Then, word clouds for both positive and negative attributions were generated using the WordCloud library.

### 3.3.2 UMAP for visualizing Fine-tuned Embedding Space

After fine-tuning the BERT model, researchers utilized the embedding of the [CLS] token, positioned at the start of each input sequence, as a representation for sequences in the labeled standard-requirement dataset. This token's embedding, derived from the model's last hidden state and through the self-attention mechanism of the transformer architecture, effectively captures the context and semantics of the sequence, performing well in aggregate representation tasks (Devlin et al., 2018b).

Subsequently, the researchers employed Uniform Manifold Approximation and Projection (UMAP) plots to demonstrate the differences in manifold representation between the pre-trained and fine-tuned models. Upon completing the model's training, the tokenizer and model weights were saved, enabling the re-instantiation of the SRC model for embedding the dataset's contents, as described in Tables 3.2 and 3.1. UMAP, a technique for dimensionality reduction, models the high-dimensional embeddings in a graph, projecting this graph into two dimensions while aiming to preserve the original topological structure. The parameters crucial for UMAP projection's repeatability and its documentation are provided in Table 3.6, which were instrumental in creating Figures 4.2 and 4.3, as detailed in the results section.

Table 3.6: SRC Task UMAP Parameters

| Parameter | Selection |
|---|---|
| n_neighbors | 1000 |
| min_dist | 1.00 |
| metric | cosine |
| random_state | 44 |

## 3.4 Standard-Requirement Linking

This section details the methodology for the standard-requirement linking task, which is the primary task performed by the researchers to answer research question two of this thesis. As a reminder, the reader may refer to Section 3.1.2 for details surrounding input and database contents used in this experiment.

### 3.4.1 Executing Linking Task

Figure 3.1 presents a visual depiction of the standard-requirement linking task. The randomly sampled project requirements are sent in iteration as queries to the Pinecone vector store, populated with standard sub-clauses as detailed in Table 3.2, with the top parameter assigned as three. Therefore, for each inputted requirement the three most relevant or nearby standard sub-clause items (as deemed relevant by Pinecones ANNS) are outputted. The standard sub-clauses were embedded utilizing ADA-002 embeddings as well as the sampled project requirements used for this experiment. The experimental framework for this task is

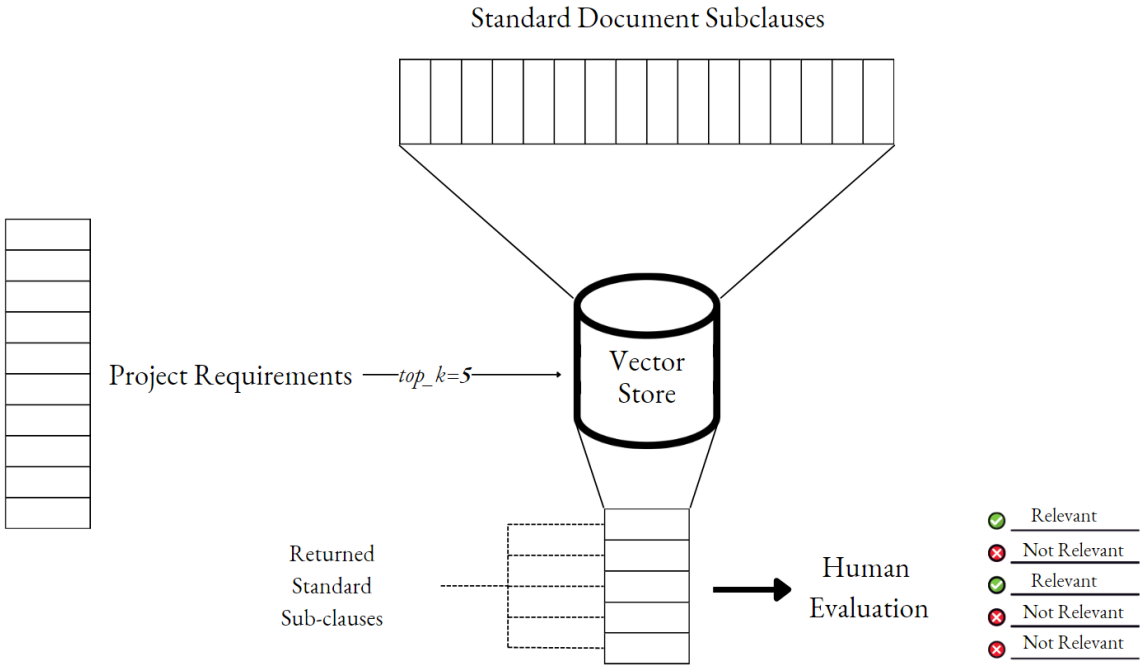


Figure 3.1: Standard-Requirement Linking Depiction

derived from the premise of requirements management software frameworks, which host project requirements in a hierarchical, structured database. In this setup, each requirement is represented as a single entity with additional attributes. These attributes often include an integer ID (often representing parent/child relationships) and the textual content or description of the requirement. Moreover, requirement entities often contain traceability links that model the requirement's relation to other requirements in the project. This framework is designed to receive only the textual content of the requirement entity as a model input. Furthermore, the model output adds a valuable attribute to the requirement entity through the creation of parent/child and horizontal linking states between project requirements and engineering standards as depicted in Figure 2.1. This facilitates the creation of verification structures, such as verification matrices or other models, for standard compliance. Similarly, project documentation and standard documentation require that, to be in compliance with a specific standard document, all standard requirements or sub-clauses must be met.

### 3.4.2 Human Evaluation

For the evaluation of language embeddings in the task of linking standards to requirements, the inputs and outputs of the model were assessed pairwise by nine human annotators. These annotators were instructed to evaluate each pair of requirements and standard sub-clauses as either relevant or not relevant. Furthermore, a definition of relevance was provided to the annotators before the evaluation session commenced. The question posed was, 'Does this sub-clause enhance or provide valuable additional information to the requirement?' The group of annotators consisted of seven graduate engineering students and one undergraduate engineering student. The evaluation survey comprised 90 items, and the sequence of the survey for each participant was randomized to mitigate fatigue effects on the respondents' results.

The respondents' results were filtered for outliers before the calculation of the model performance. The high level of technicality present in both the project requirements and the standard sub-clauses combined with varied levels of previous knowledge caused some pairs to lack sufficient agreement to be considered for processing of the results. The researchers utilized the majority rank of the two response classes

for each requirement- standard link to calculate the performance of the model, represented as percentage relevancy. Majority rank and human evaluation have been shown to be sufficient methods of evaluating semantic search frameworks (Elbedweihy et al., 2015).

## 3.5 Standards Search Evaluation

To extensively assess the effectiveness of the embedding-based standards search framework, a practical case study was conducted using an industry validation dataset. The study involved a collaboration with a product designer, who supplied the research team with the standard documents employed in their design methodology as well as a conceptual description of the product. This documentation underwent preprocessing before its integration into our vector store database, culminating in the addition of 89 distinct chunks. The investigation then proceeded to evaluate three distinct standard document retrieval approaches, employing the documents supplied by the product designer as a benchmark for precision. The methodologies implemented for evaluation were as follows:

1. ANNS
2. ANNS with Cohere Rerank
3. ANNS with GPT Zero-shot Reranking

The first stage of retrieval in all three methods is the Approximate Near Neighbor Search (ANNS) technique further detailed in Subsection 3.5.1. The second method evaluated by this research combines ANNS with a state-of-the-art reranking API endpoint offered by Cohere. Then lastly, a GPT Zero-shot reranking approach was tailored to this task and evaluated. Figure 3.2 illustrates an abstraction of the reranking approach of our embedding-based standards search.

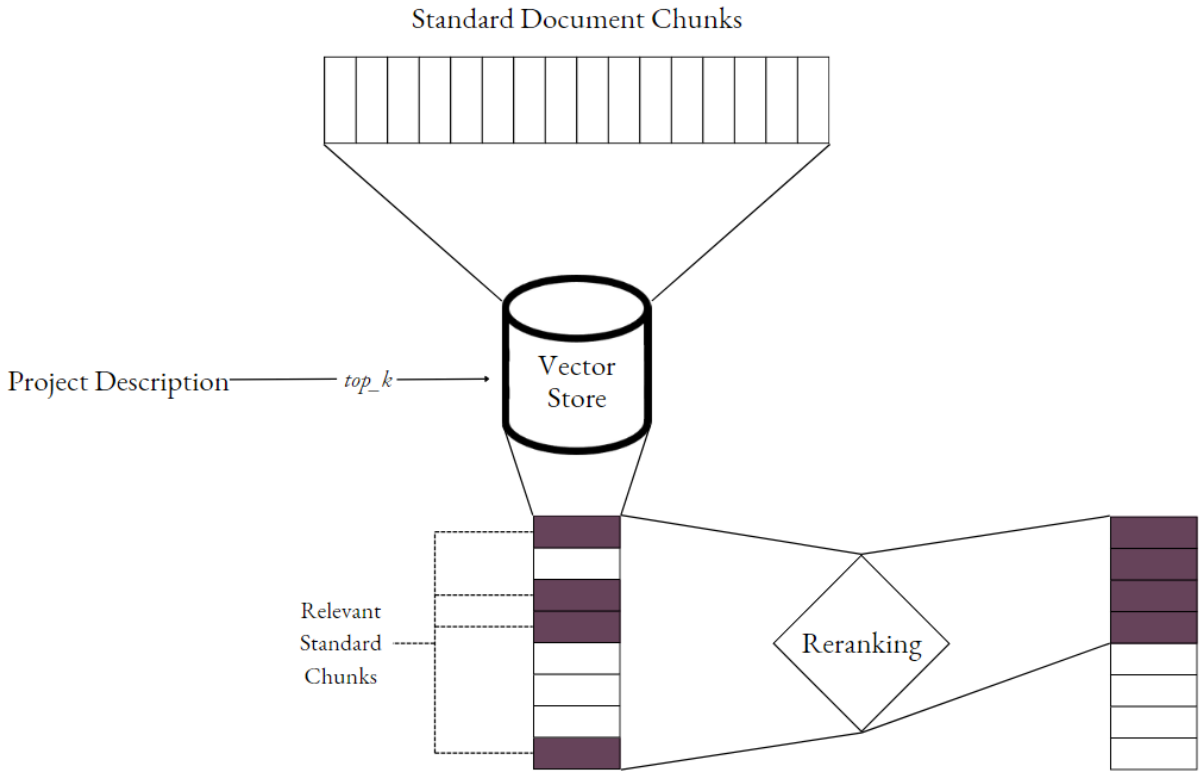


Figure 3.2: Reranking of Search Results

### 3.5.1 ANNS

To investigate ANNS for extracting relevant standard documents from the vector store database, Pinecone's query API endpoint was utilized. This allowed the researchers to host the embedded standards in cloud storage and interaction through API query calls. While Pinecone's exact algorithms are proprietary, it is known that typical ANNS methods are used. Most of the optimization done by Pinecone is expected to be done to reduce latency as a client for the searches, especially in very large datasets.

Following the initial embedding of the product description via the text-embedding-ada-002 model, the system extracted relevant standard documents, also embedded using the same model. This process was facilitated by Pinecone's query interface, which allows for the specification of the number of document identifiers to be retrieved, prioritizing standard documents closest in the vector space as determined by

Pinecone's Approximate Nearest Neighbor Algorithm. By employing strategies such as indexing, clustering, hashing, and quantization, ANNS algorithms significantly narrow down the search space and enhance lookup times, thereby improving both computation and storage efficiency at the expense of some accuracy.

The essence of ANNS's efficiency lies in its ability to focus on the true intrinsic dimensionality of the data, striking a delicate balance between accuracy and performance. This method is particularly beneficial for applications requiring similarity search, where the objective is to find items similar to a given query item based on vector representations. In such contexts, the slight loss in precision due to the use of approximate methods is usually outweighed by substantial gains in speed and scalability. Pinecone, while specifics are proprietary, utilizes Locality Sensitive Hashing (LSH), indexing, clustering, and quantitation.

### 3.5.2 ANNS with Cohere Rerank

After evaluating the semantic search utilizing just the Pinecone Query endpoint, the research team applied Cohere's Rerank API to the initial retrieval results, utilizing Rerank as a cross-encoder to assess pairwise relevance between the project description and each document chunk retrieved. This then yields a list re-indexed according to Cohere's relevance metric, designed to rank the documents from most to least semantically relevant to the query or product description. As a reminder, in the context of this study, this means that the standard document segments most pertinent to the project description will be ranked highest.

### 3.5.3 ANNS with GPT Zero-shot Reranking

Finally, a GPT zero-shot approach was evaluated as the reranking agent. Utilizing the following $Classification\ Prompt$, each first-stage retrieval result was passed in iteration to the OpenAI's Completions endpoint.

- $CLASSIFICATION\ PROMPT$ = " You are an Assistant responsible for helping detect whether the retrieved engineering standard is relevant to the project description. For a provided

input, you need to output a single token: "Yes" or "No" indicating the retrieved standard is relevant to the project description query."

Additionally, the model was limited to only returning a binary classification by setting the $logitbias$ model parameter to the logit values for "Yes" and "No" and assigning $maxtokens$ as 2. While this alone would return a list of relevancy-filtered engineering standards, the ranking of the index is derived from the $logprobs$ output from the endpoint. This returns a logarithmic probability of the model's assignment of either "Yes" or "No" in the response. This allowed the researchers to then rank the list based on this probability, after conversion to non-logarithmic probability. The higher the probability of a "Yes" classification, the more relevant the engineering standard is considered to be for the project description. Therefore, the final ranking of retrieved engineering standards is directly derived from these probabilities.

### 3.5.4 Industry Validation

For the three approaches that the researchers evaluated, the benchmark case was formed from an industry partner that provided the researchers with the standards that were used in the development of one of their recent production products as well as a conceptual description of the product. As a reminder, in this task, the standard documents identified by the industry partner were chunked and inserted into the vector database alongside all other documents as detailed in Table 3.4.

## 3.6 Standards Augmented Requirements Generation (SARG)

The Standard Augmented Requirements Generator (SARG), is a special adaption of RAG that was developed to synthesize design information and constraints contained in relevant engineering standards documents. This information is then presented to the user as high-level requirements that summarize the relevant topical requirements from the document and include citations to the source document for later detailed reference by the designer. The GPT zero-shot re-ranking approach was utilized as the retriever for the SARG pipeline, based on results from evaluations presented later in this paper.

### 3.6.1 Model Architecture

The use of the GPT Zero-shot augmented search allowed for only the responses deemed relevant (assigned "Yes") to be passed to the generation component of SARG. Then, these relevant retrieved standard document chunks are passed in iteration to the GPT Completions endpoint, along with the project description and the $GENERATION\ PROMPT$. All prompts and code utilized for this framework may be found in Appendix H. In summary, the $GENERATION\ PROMPT$ asks the model to survey the standard retrieval chunk and extract the contextually relevant information while maintaining citation to the source document. This produced an output of N text files, where N is the number of retrieved and reranked results that are passed to the generation function. The output of the generation function is then compiled into one text file for processing and compilation by the compilation function. The model is prompted again, with the $COMPILATION\ PROMPT$, which tasks the model with eliminating contextual redundancy of the generated requirements and logically organizing the requirements based on the project description. Lastly, the condensed requirements are passed to the model once again, with the $VERIFICATION\ PROMPT$, which is tasked with ensuring that proper requirements syntax is used and once again verifying completeness, by comparing the output of the compilation step, the generation step, and the project description.

### 3.6.2 Model Evaluation

The SARG tool was evaluated by for relevancy and compliance by the product owner used for the test case. In addition, qualitative observations by the researchers are also provided for the generated requirements. This provides multiple perspectives on the generated requirements in that both the model's ability to source and synthesize relevant information is accessed but also the syntactical quality in which it generates requirements in this framework. Previous work on LLM-generated content describes industry expert evaluation as a useful tool for assessing content (Jury et al., 2024). For this evaluation the inventor of the product used to prompt SARG, an industry expert with more than 30 years of experience in engineer-

ing and design, was asked to evaluate auto-generated requirements using the metrics represented in 3.7. The metrics were chosen from key features used to specify good requirements (Firesmith, 2003). The percentage of requirements that passed the evaluation was then observed and discussed. In evaluation of

Table 3.7: SARG Industry Case Metrics

| Metric | Prompt |
|---|---|
| Relevance | Is the requirement applicable to the product? |
| Compliance | Does the product meet the requirement? |

compliance, the product owner further surveyed the elicited requirements in observance of their products conformance to guidelines or criteria present in the requirements.

# Chapter 4

# Results

Section 4.1 presents the results of the fine-tuned BERT model on the standard-requirement classification task. In addition, Section 4.1 provides analytical results as yielded by the Integrated Gradients method to further address research question one. Then, the results for the standard-requirement linking task are presented in Section 4.2 along with UMAP projections visualizing the standard-requirement corpus latent space. Then, Section 4.3 presents the efficacy of utilizing embeddings to help bridge the current information retrieval gap existing with engineering standards. Finally, Section 4.4 presents results of an outcome of this research, the SARG tool.

## 4.1 BERT Fine-tuned for SRC Task

Table 4.1 shows the results obtained by training the BERT model on the SRC task. The training loss decreases across each epoch of training as the model effectively learns the classification task. After the third epoch, the early stopping function is triggered (by a decrease in MCC of greater than 0.01) and the model completes training. The validation loss, which is representative of how well the model is generalizing on unseen data, slightly increases in the third epoch signifying that if further training is completed generalizability will suffer. The MCC score of 0.901 for the last epoch indicates that the model is performing well on the validation dataset. This performance is further evaluated using the testing dataset.

Table 4.1: SRC Task Training Metrics

| Epoch | Training Loss | Validation Loss | MCC |
|---|---|---|---|
| 1 | 0.474 | 0.187 | 0.883 |
| 2 | 0.134 | 0.099 | 0.914 |
| 3 | 0.042 | 0.105 | 0.901 |

Table 4.2 illustrates the BERT model's efficacy on the testing dataset through various performance metrics. Precision represents the model's accuracy in identifying true positives among all positive predictions, highlighting its ability to correctly distinguish standards and requirements without incorrectly predicting assigning class levels. Recall represents the model's capability to capture all relevant instances within the dataset and is representative of the division of model-assigned class positives and ground-truth positives in the dataset. F1-Score combines precision and recall into a single metric through the calculation of the harmonic mean of precision and recall thus providing a comprehensive measure of the model's overall accuracy and reliability in classifying standards and requirements accurately. Support refers to the number of ground-truth occurrences for each class in the dataset and is representative of the size of the unseen dataset for the model.

Table 4.2: SRC Task Result Metrics

| Label | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| Requirements | 0.97 | 0.96 | 0.96 | 255 |
| Standards | 0.95 | 0.96 | 0.96 | 201 |

The confusion matrix displayed in Figure 4.1 visualizes the fine-tuned model's performance on the test dataset. In review of methodology, 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 255 requirements in the test dataset, the model correctly labeled 245 for a recall of 96 percent. Furthermore, out of the 201 standards in the test dataset, the model correctly labeled 193 for a recall of also 96 percent. The overall performance of this model on the test dataset may be summarized by the MCC score of 0.92. This indicates that there exists a high correlation between predicted labels and the actual labels present

in the dataset. These results indicate that a fine-tuned BERT model may be used to distinguish between engineering requirements and standards.

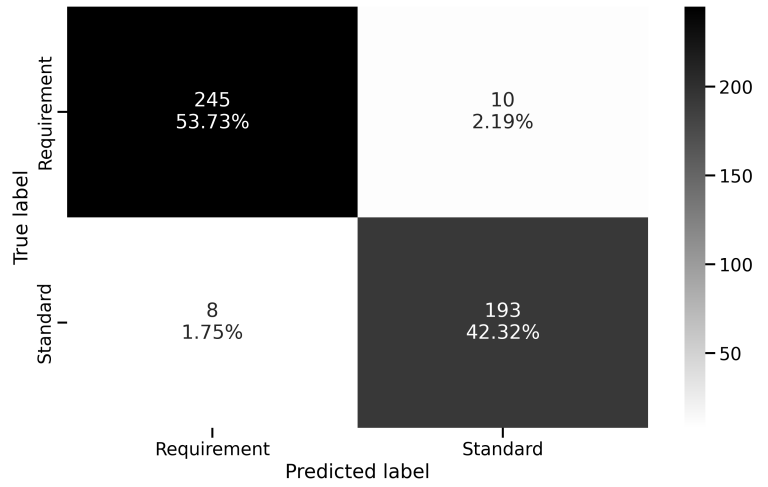


Figure 4.1: Standard-Requirement Classification Task Confusion Matrix

UMAP manifold projections for the pre-trained and then fine-tuned BERT models are illustrated in Figure 4.2 and Figure 4.2. Parameters used for the creation of these projections may be found in Table 3.6. Figure 4.2 demonstrates the topology of the standard-requirement classification training dataset, as detailed in Tables 3.1 and 3.2, with standard sub-clauses represented by red plot items and project requirements by black plot items. The data points for both classes occupy a continuous and overlapping space on the latent manifold, indicating a gradual transition between the classes, which supports the validity of the training dataset for the evaluation of a binary classifier. Additionally, the lack of pronounced cluster formation suggests that the model, when processing sentences or small paragraphs, does not cluster project requirements or engineering standards based on low-level features such as syntactical occurrences or parent document relationships. This observation suggests to researchers that the latent representation from the pre-trained BERT model is capable of capturing high-level features. However, this projection also implies that while the model is capturing context or semantics, it has not yet formed distinct clusters for requirements and standards, indicating that the base embeddings have not achieved a sufficient level of

abstraction to discriminate between these classes on the latent manifold. This suggests to the researchers that the latent manifold distribution is driven by topical or contextual arrangement.

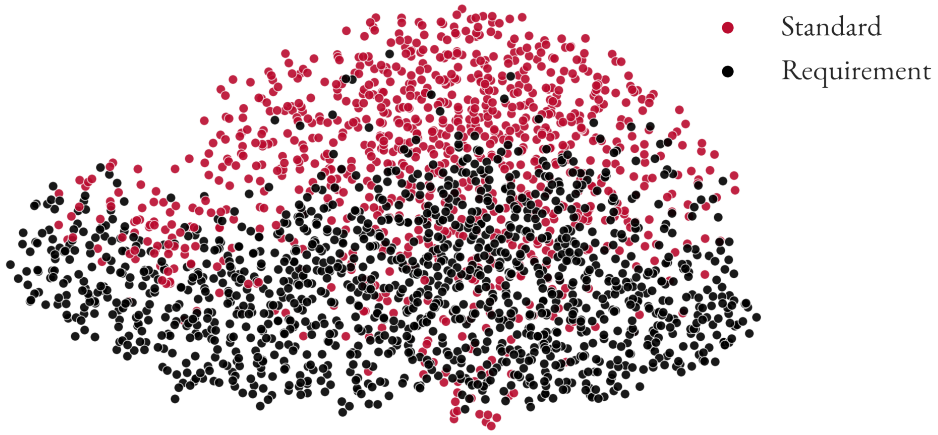


Figure 4.2: Standard-Requirement Manifold Before Fine tuning

Figure 4.3 illustrates the BERT embedding space after fine-tuning. Two distinct and distant clusters representing the classification of standards and requirements have formed, visually indicating the results detailed in Table 4.1. This projection provides visual confirmation of the fine-tuned BERT model's ability to correctly classify standards and requirements. Along with a qualitative analysis of the preceding graphic, Figure 4.2, Figure 4.3 suggests that the model's latent representations have reached a higher level of abstraction, offering a more generalized distinction between requirements and standards.

To gain further insight into the black box performance of the fine-tuned BERT model, the researchers applied the integrated gradients technique to the fine-tuned model as detailed in Section 3.3. Figure 4.4 illustrates the frequency, or number of words, that exist across different attribution values. As hypothesized by the researchers, the distribution is Gaussian in nature, indicating that a majority of the words present in either standards or requirements are somewhat common in attribution among the classes. This indicates that the model is not reliant on a small subset of keywords but on the contextual and semantic layering of common engineering terminology to correctly predict classes. The mean attribution of the analysis is situated close to zero, further indicating that specific words or subsets of words do not bias the model towards the prediction of a class unless the justification is enhanced by the context.


Figure 4.3: Standard-Requirement Manifold After Fine-tuning

Figure 4.5 illustrates the word cloud generated from the requirement class token attributions. This analysis reveals a lexicon consisting of verbiage that one would anticipate in procedural and action-oriented documents such as requirements documentation. Prominent terms in this depiction such as "submit," "include," and "fabrication," are suggestive of the dynamic processes inherent to engineering requirements. Furthermore, these tokens are characteristically functional in nature, implying steps or components necessary for the fulfillment of engineering tasks. Terms such as "planning" and "installation" reinforce this characterization by highlighting the sequential or methodical aspects detailed in required in engineering requirements that guide engineering projects from inception to completion. Direct verbs, such as “submit,” “include,” and “write” highlight the imperative tone found in requirements statements.

Figure 4.6 illustrates the word cloud generated from the standard class token attributions. The absence of directive verbs typically associated with requirements highlights a more declarative tone consistent with the descriptive nature of standards. Terms such as "specification", "procedures", and "verification" indicate that this class of language is not characteristically prescribing actions but is delineating expectations and definitions.

Modal verbs often found in requirements documentation such as shall, will, must, or should held attribution weights of -0.184, -0.619, -0.2682, and -0.170 respectively. Thus, indicating that overall the

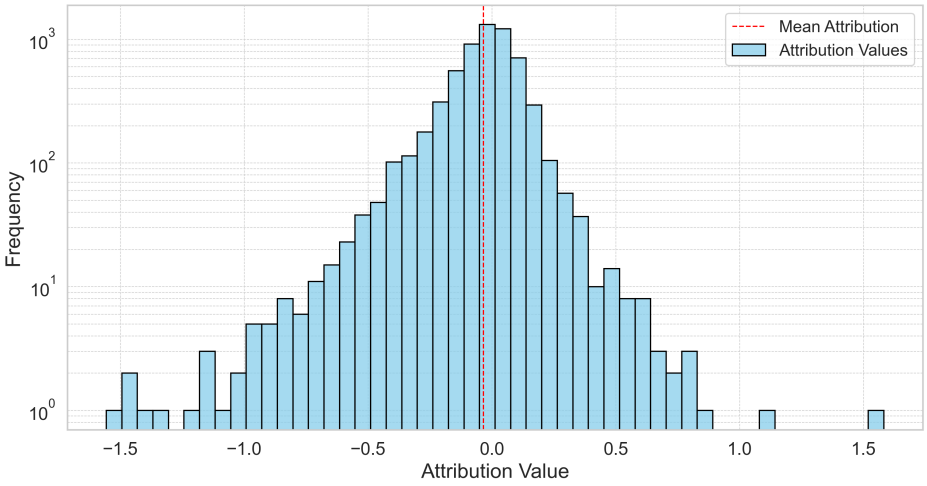


Figure 4.4: Histogram of Fine-tuned Model Word Attributions

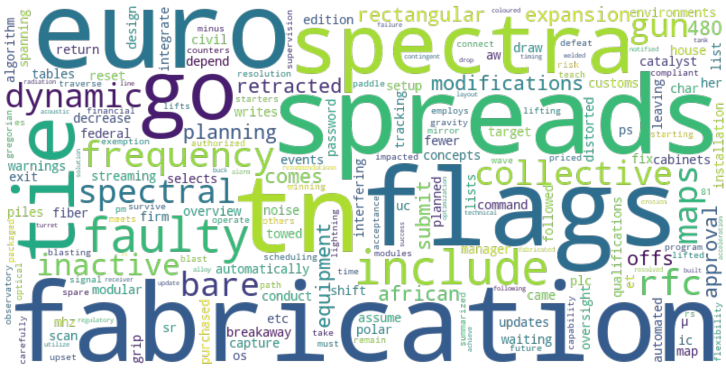


Figure 4.5: Word Attribution Analysis Highlighting Key Determinants of 'Requirement' Classification

model utilizes the presence of these words to predict the requirement class. Selected word attributions are shown in Table 4.3. Wh-pronouns such as who, what, and whom are more indicative of standard prediction and the researchers attribute this to the descriptive nature of standard documentation, detailing guidelines for multiple organizational systems in contrast to the nature of requirements specifying guidelines for one system or component. Modal verbs such as may, might, can, and could are more indicative of standard prediction while modal verbs such as would, should, shall, must, and will are more indicative of requirement prediction. Additionally, the token "s", a possessive ending, is more indicative of standard prediction with an attribution of 0.110. The possessive ending, "s", is indicative of standard prediction in


Figure 4.6: Word Attribution Analysis Highlighting Key Determinants of 'Standard' Classification

which guidelines are specified for entities while requirements are singular in nature and assign targets or goals to one entity.

Table 4.3: Selected Word Attributions

| Wh-pronouns | Attribution | Modal Verbs | Attribution |
|---|---|---|---|
| Who | 0.223 | May | 0.215 |
| What | 0.154 | Might | 0.068 |
| Whom | 0.086 | Can | 0.044 |
| | | Could | 0.025 |
| | | Should | -0.017 |
| | | Shall | -0.184 |
| | | Must | -0.268 |
| | | Will | -0.620 |

## 4.2 Efficacy of Embeddings for Standard-Requirement Linking

This section details the results of the standard-requirement linking task. The methodology for this task is detailed in Section 3.4 while relevant statistics for the corpus may be referenced in Subsection 3.1.2.

Figure 4.7 illustrates the raw results of human relevancy evaluation for Project 1. Project 1 details the design requirements for a threaded pipe manufacturing production line. After analyzing the results at the Project level, the S2R method achieved 50.0 % relevance accuracy.

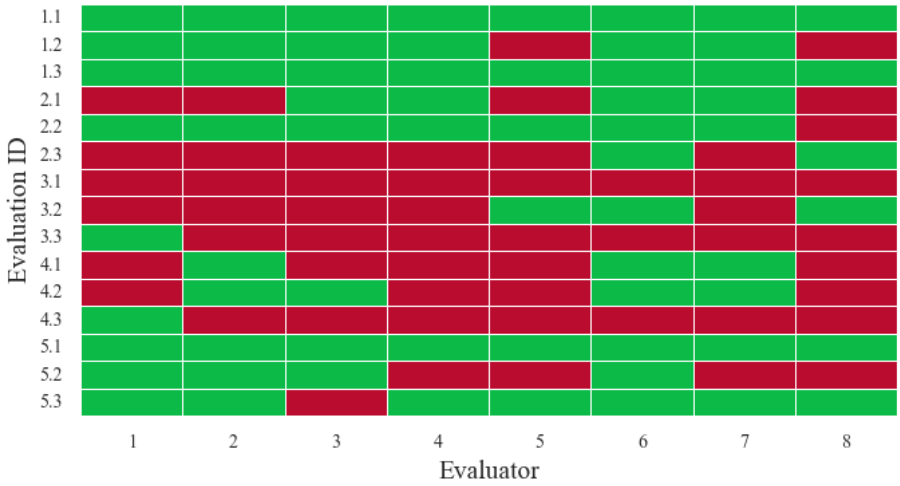


Figure 4.7: Project 1 Evaluator Results

Figure 4.8 illustrates the raw results of human relevancy evaluation for Project 2. As a reminder, Project 2 focuses on design requirements for manufacturing stations and equipment for exhaust gas flaps. For the Project requirements sampled from Project 2, the S2R method achieved 64.29 % relevance accuracy.

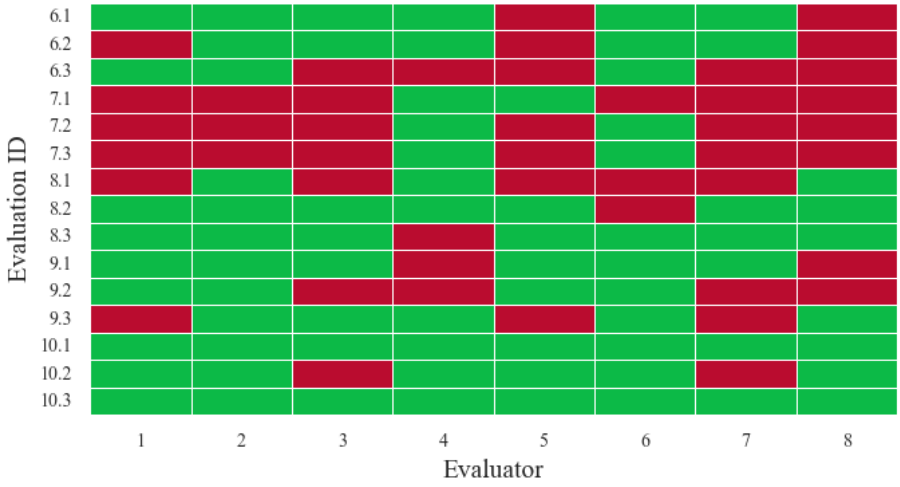


Figure 4.8: Project 2 Evaluator Results

Figure 4.9 illustrates the raw results of human relevancy evaluation for Project 3. Project 3 enumerates the design requirements for industrial textile manufacturing equipment. Across the projects sampled and evaluated, the S2R method performed second-best on this Project with a relevance accuracy of 69.23 %.

Figure 4.10 illustrates the raw results of human relevancy evaluation for Project 4. Project 4 describes

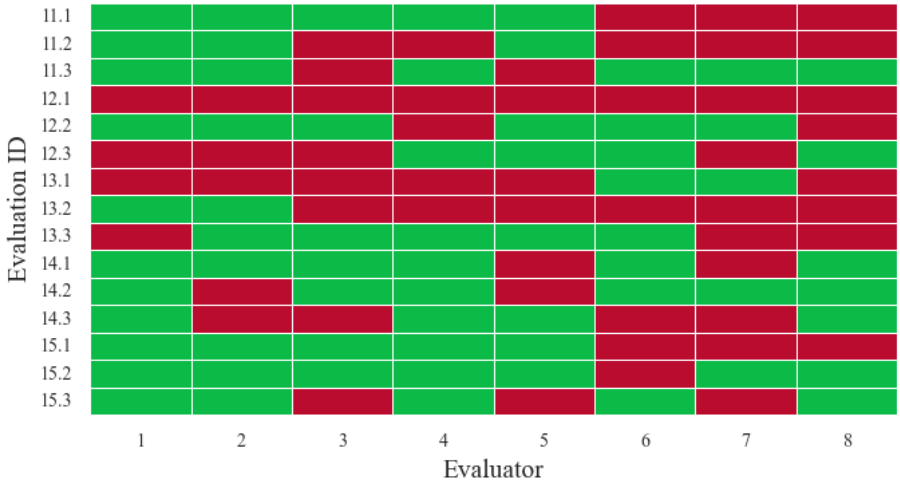


Figure 4.9: Project 3 Evaluator Results

the design requirements for a material handling system in the pipe threading manufacturing process. For this Project, the S2R method achieved 36.36 % relevance accuracy, the second lowest of the projects evaluated. Figure 4.11 illustrates the raw results of human relevancy evaluation for Project 5. Projects 5

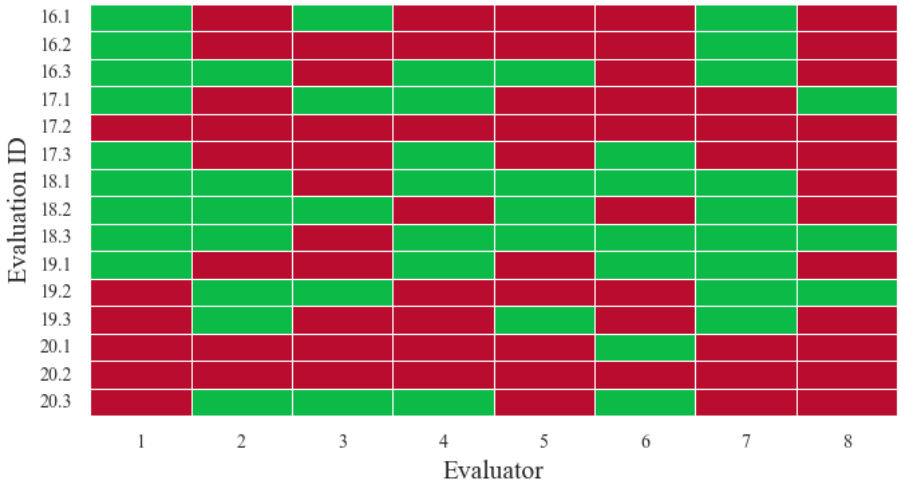


Figure 4.10: Project 4 Evaluator Results

and 6 consist of publicly available requirements for design subsystems within the Square Kilometre Array (SKA) Project. Specifically, Project 5 describes the design of a dish element, while Project 6 describes an

artifact responsible for correlating and beamforming. While these two requirement sets are derived from the same system, there is little overlap between the requirements. Project 5 makes no mention of the correlator and beamformer described in Project 6. Furthermore, a survey of Project 6 for the dish element detailed in Project 5 reveals that it was mentioned in only 4 requirements.

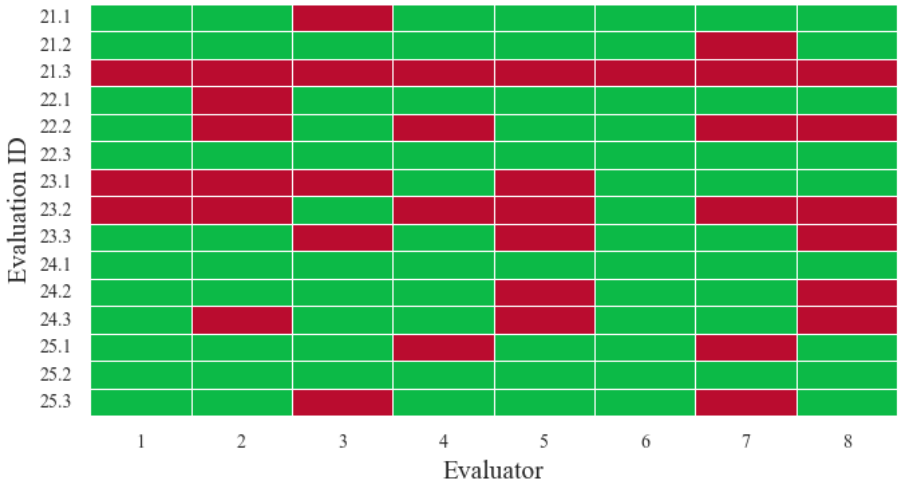


Figure 4.11: Project 5 Evaluator Results

Figure 4.12 shows the raw results of human relevancy evaluation for Project 6, which describes an artifact responsible for correlating and beamforming. While Project 5 attained a relevance accuracy score of 84.62%, the highest among the projects, the relevance accuracy for Project 6 was calculated as 33.33%, the lowest relevance accuracy in the group. Project 6 is unique from the other documents structurally in that every requirement begins with the product's name, "CSP_Mid.CBF," and other projects only contain the product name occasionally, and in varied positions in the sentence.

After filtering out evaluation pairs from the analysis that did not show majority agreement on relevancy assignment, the overall performance of the model is presented with an 81.2% relevance accuracy. This suggests to the researchers that this method can be utilized with some success for dynamically linking engineering standards to engineering requirements.

Figures A.2 and A.1 present UMAP projections of the latent manifold as yielded by language embeddings. Figure A.2 offers a broad view of the latent manifold, reinforcing key findings from the analysis

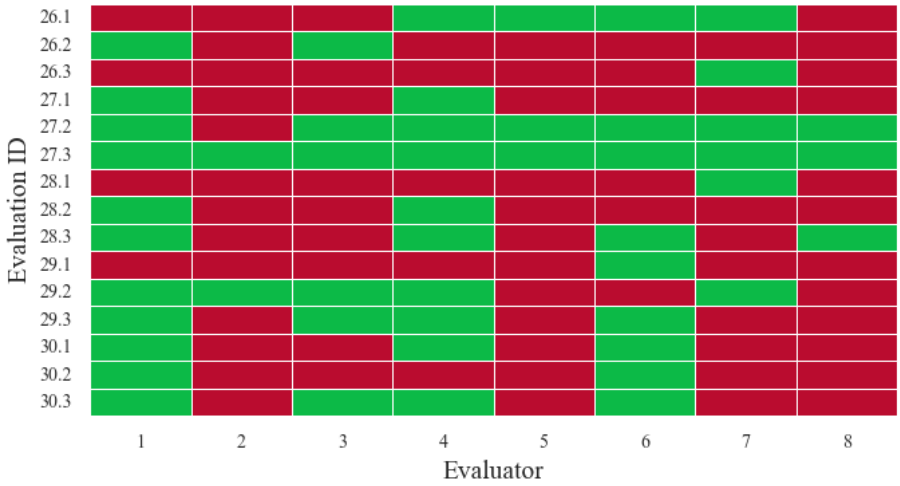


Figure 4.12: Project 6 Evaluator Results

using BERT in the previous section. Primarily, it shows that standards and requirements have significant overlap on the latent contextual manifold when plotted together, thereby validating this approach for standard-requirement linking.

Figure A.1 depicts the same data, yet labeled by origin. Standards sub-clauses originating from AASHTO, ASABE, and ASTM are observed to be more tightly clustered than other groups depicted on the manifold. Furthermore, Project 1 appears distinct from the other data on the right side of the projection, with little overlap with other items. A noticeable cluster exists in the left-center of the depiction, comprised of standard sub-clauses from CCSDS, which provides standards for space data communication and thus hosts significant differences from other items in the dataset.

The framework's performance on the testing dataset, alongside a qualitative analysis of the latent manifold, indicates that this framework may be effective for the task of linking standard sub-clauses to requirements.

## 4.3 Efficacy of Search Framework with Document Chunk Embeddings

Figure 4.13 illustrates the outcomes of the standard retrieval task. To clarify, precision within the industry validation case refers to whether the extracted segment comes from documents deemed relevant by the product owner. If so, that specific outcome is considered relevant, thereby contributing to the measure of precision. It's important to note there may be relevant standards within the retrieved items that were not included in the product owner's initial specifications. Nevertheless, to avoid subjectivity, such possibilities are not reflected in the results. Instead, only the 'ground truth' as determined by the product owner is included.

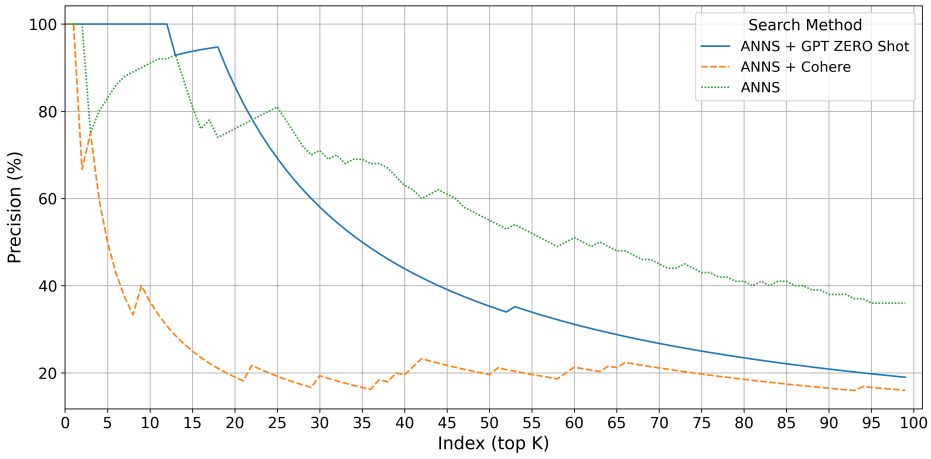


Figure 4.13: Evaluation Results of Search Methods

The graph in Figure 4.13 illustrates the varying precision levels of three semantic search methods as the retrieval index (k) increases. The ANNS semantic search method exhibits a steep decline in precision from k=0 to k=5, indicating that the quality of results drops quickly as more items are retrieved. This method experiences a brief improvement at k=13, matching the GPT zero-shot method's performance before continuing a general decline towards k=100. The noticeable fluctuation in precision early in the graph is caused by the introduction of an irrelevant result at k=4. In contrast, the ANN semantic search with

Cohere Rerank demonstrates inconsistent precision across the range, with a notable decrease to below 20 percent precision by k=20. This suggests that this method struggles to appropriately correlate the project description with relevant standards. The GPT zero-shot reranking method, however, maintains a 100 percent precision rate until k=13, at which point an irrelevant result is introduced. It's important to note that this method evaluated only 15 results as relevant, and these are predominantly placed within the top 13 results, explaining the smooth and steep decline in precision thereafter. In comparison, while the ANNS method begins to surpass the GPT zero-shot method in precision after k=23, the initial retrieval (such as the top 20 results) is significantly more precise with the GPT zero-shot approach. This indicates that the GPT zero-shot method is more effective at ranking the most relevant results at the top of the retrieval list. The behavior of these search methods is further explored by the use of the Uniform Manifold Approximation and Projection (UMAP) dimensionality reduction technique. The initial first stage retrieval, ANNS-based search as executed by Pinecone Query, top 100 results are visualized in Figure C.1. The darker elements represent the retrieved results while the lighter elements depict the manifold projections of the standards in the vector store.

Figure C.2 visualizes the results of the top 100 results, derived and reranked from an initial result of 500 by the ANNS-based search, and the result's position on the manifold. Notably, this method utilizing Cohere Rerank illustrates distinct clustering of results.

Figure C.3 highlights the manifold plotted results of the GPT Zero-shot method and illustrates a distinct and centralized cluster of results, visualizing GPT's ability to filter the contextually relevant documents to the conceptual project description. This depiction further validates the semantic search approach as valuable, demonstrating that engineering standards relevant to a project exist in a unique space on the contextual embedding manifold, and this space may be accessed through the embedding of a conceptual description. In summary, ANNS first stage retrieval followed by GPT zero-shot reranking demonstrates exciting performance for filtering contextually relevant engineering standards provided only a conceptual project description as an input. The poor performance of Cohere Rerank is attributed to its training

on primarily question-answer tasks, which may not apply to the task of this work. Based on its superior performance in this task, the GPT Zero-shot method was chosen for integration into the SARG pipeline.

## 4.4 SARG Results

The industry product owner surveyed the SARG-generated requirements and deemed 64 percent of the generated requirements as relevant to the product and identified the product as compliant with 50 % of the generated requirements. Example generated requirements are detailed below with the specific product name and standard references filtered to protect proprietary information.

- Springs will have an average load rate within ±10 % tolerance for springs with five or more total turns, calculated as per [standard] S6.1.
- Tolerance on the load rather than the height dimension of springs will be between 20 and 80 percent of calculated solid capacity, not less than 65 % of the spring's calculated solid capacity, as per [standard]
- The [product name] will have all mechanisms designed with no protrusions in the direction of rotation and provide PTO drive shaft guarding as specified in [standard] S5.5.1 and S5.5.2.
- Safety signs will be placed on the PTO drive shaft guard, outer driving member of PTO drive shafts, and prominently to indicate the necessity of keeping guards in place, conforming to [standard] as per [standard] S6.1 to S6.4.
- Guards for moving parts will minimize the possibility of inadvertent contact, requiring tools for removal, and be designed to withstand a perpendicular static load of 1,200 N without permanent deflection as per [standard] 10.2.1 and 10.2.3.
- The [product name] will incorporate a secondary power input connection (SPIC) and a PTO drive shaft that meets the closed and extended length requirements, utilizing a universal joint or a

wide-angle constant velocity universal joint if operation angles exceed 50°, as specified in [standard] Sections 3.2.4 to 3.6.

- A non-rotating PTO drive shaft guard and a PIC [SPTO] [SPIC] guard will be equipped, fully covering and enclosing the PIC, SPTO, and SPIC, ensuring comprehensive protection as per [standard] Sections 3.9 to 3.11.

- The [product name] will be equipped with a rigid imperforate guard on the top and sides, meeting the requirements of the foot probe test and the thrown-object test as outlined in [standard] to ensure comprehensive protection.

In the follow-up observations shared by the product owner, it was discovered that the initial results showed a relevancy of only 64%. This underestimation was found to be due to giving the responsibility for meeting specific spring standards to the company's spring supplier. After taking this factor into account, the true relevance of the requirements was adjusted to 100%. Additionally, when assessing the product's compliance, the product owner initially rated it at 50% compliance. However, upon closer examination, the researchers found that the product was found in compliance with all but one of the initially relevant requirements identified by the SARG process. The compliance metric was developed to capture important information about the product, specifically looking for gaps in engineering documentation management or other common issues companies face in meeting standards, often due to a lack of traceability in compliance. Furthermore, the researchers acknowledge the need for broader testing to more accurately determine the SARG model's effectiveness throughout the product development process, especially in compliance verification and validation.

Overall, the model succeeded in capturing and synthesizing relevant information from the retrieved engineering standards in a concise, requirements-structured manner. This enables designers to easily constrain the design space to comply with standards while allowing for the natural development of product form. Furthermore, the SARG tool can assist designers by ensuring that design recommendations from engineering standards are realized early in the design process, potentially reducing iterations and costs.

Moreover, this implementation demonstrates the competence of Large Language Models (LLMs) in retrieving and synthesizing relevant engineering standards, as well as eliciting tailored high-level project requirements.

# Chapter 5

# Discussion

This chapter presents the dicussion of results of this thesis. Section 5.1 frames the result of this work to answer research questions. Section 5.2 presents the expected impact on design research and practice of this work. Finally, Section 5.3 presents key recommendations derived from this work. Limitations for this thesis are then presented in Section 5.4.

## 5.1 Addressing Research Questions

**RQ 1** - How may requirements and standards be distinguishable through large language models?

The high performance of the fine-tuned BERT model in the standard-requirement classification task proves that requirements and standards may be distinguished by the use of language embeddings effectively, highlighting the model's capability to discern nuances in standards and requirements that may elude manual detection. Moreover, this indicates that requirements and engineering standard sub-clauses are sufficiently distinct for the model to leverage the differences in the syntactic and semantic view to perform well on unseen data. As a reminder to the reader, the training dataset used for this project contained data points from six different industry requirement sets and ten unique standard-developing organizations. The heterogeneity of this dataset, while not claimed as perfect, leads the researchers to have high confidence that this may serve as a generalized approach that may be leveraged in design practices for

detecting the explicit presence of standard line items in requirement sets. This brings value in the direct appearance of a prescriptive standard sub-clause in requirement sets that may be flagged, allowing for either manual insertion of incorporation by reference or for further automated links as proposed in the standard-requirement linking task. Further augmenting the standard compliance traceability as well as pathways for requirement reuse selection by parent standard selection for a project.

The blending of these data points on the latent manifold indicates a smooth transition between the two classes, validating the training dataset's appropriateness for a binary classification task. The absence of distinct clustering suggests that the model does not rely on low-level features like syntax or document origin for manifold but rather on high-level features captured by the pre-trained BERT model's latent representations. However, the lack of clear separation or clustering between standards and requirements also suggests that the model's base embeddings need further refinement to adequately distinguish between these two classes based on their inherent characteristics further validating the need for fine-tuning the BERT model.

The application of Integrated Gradients to the fine-tuned model and the plotting of term frequency vs word attribution yields significant understanding in how requirements and standards may be distinguished through large language models. Additionally, by the analysis of the word attributions the researchers were able to verify that the model's ability to accurately classify information is not based on a limited set of specific or rare keywords. Instead, it relies on the contextual and semantic structuring of commonly used engineering terms. The average attribution of the analysis is near zero, further showing that the model is not swayed toward classifying any particular category based on certain words or groups of words unless the context substantially supports the reasoning.

**RQ 2** - How may language embeddings correlate engineering project requirements to standard document sub-clauses?

Due to limited data availability, the experiment for linking standards to requirements through language embeddings, which shows promise, could not be conducted under ideal conditions. The approach demonstrated reasonable accuracy in identifying relevant standard sub-clauses or guidelines to link with

requirements. However, this accuracy is constrained by the narrow context of individual line item requirements and standard sub-clauses. Initial impressions may suggest that the links generated are pertinent to the given requirements, assuming the reader's understanding is limited to the context provided by the model. Upon closer examination of the source documents, a frequent misalignment in scope becomes evident. Nonetheless, this limitation is mitigated in scenarios where engineering firms maintain a detailed database of relevant standard documents, suggested to be vectorized and stored as sub-clauses. This practice is common among such firms, and in these instances, having a specific set of standard documents helps to eliminate irrelevant content, ensuring only pertinent links are established.

**RQ 3** - Do language embedding correlations exist between conceptual engineering project descriptions with relevant standards? Through the evaluation of three different search models, the researchers conclude that language embedding correlations do exist between conceptual engineering project descriptions and relevant standards. This is proven by the use of an industry validation case, which provides seldom-acquired information regarding the standards used in the development of the product, as well as access to the product designer. The narrowing of the search results through the refinement of methods further indicates that relevant standards for a project do indeed occupy a localized space on the contextual manifold. Furthermore, the ability of the GPT Zero-shot approach to accurately filter out irrelevant engineering standards exhibits a high level of engineering and contextual reasoning by the model.

## 5.2 Impact on Design Research and Practice

This thesis advances knowledge in the NLP domain, exploring the application and performance of several common NLP techniques on engineering standards. These standards are domain-specific concerning common engineering terminology and reveal differences in how large language models process and perceive them. The successful deployment of GPT-Zero shot as both a reranker and a relevancy filter for engineering standards offers significant potential to enhance search processes for engineering standards and design practices. Furthermore, the domain of requirements engineering, and specifically requirements

management, is advanced by the creation and evaluation of the standard-requirement classification model and the evaluation of the standard-requirement linking framework. The standard-requirement classification model has proven to be an effective approach for identifying "stray" engineering requirements that may be placed in engineering documentation without explicit reference to a source standard, thereby enhancing compliance traceability, potentially reducing iterations and costs, and promoting product safety. The S2R framework, proposed in this work for standard-requirement linking, offers practical benefits by reducing the time spent on eliciting requirements for variant or adaptive designs within a similar solution space. Since most variant designs must comply with regulatory, safety, environmental, or design guidelines, this underscores the practicality of the approach to reduce the time spent in requirement elicitation for designs in a similar solution space. While engineering standards currently lack thorough exploration, this work aims to inspire future research in this area. Furthermore, given the scarcity of publications on engineering standards, Section 2.1 of this thesis adds value by conglomerating the key, albeit rare, references on the subject.

## 5.3 Recommendations

The primary recommendation of this work is for the benefit of Standard Developing Organizations. Enhanced methods for interacting with engineering standards have been developed in this work, combating the pain point designers often face with project-relevant standard retrieval. Furthermore, the researchers encourage SDOs to adopt practices as depicted in this thesis to not only increase the use of their hosted standards but to primarily lessen the pain point that designers currently face with current systems. Additionally, through analysis of the fine-tuned BERT model, the researchers have developed knowledge surrounding how LLMs interact with engineering standards and requirements alike. Through the analysis of term frequency and word attribution, derived from the fine-tuned model's weights, the researchers may confidently recommend that TF or TF-IDF methods will not perform well on the task completed by the fine-tuned model of highlighting the presence of engineering standards that are present without appropriate reference or linking to the source document. Furthermore, as the fine-tuned model has been

shown to not use hyper-specific terms to predominantly distinguish requirements and standards, the researchers believe that TF or TF-IDF methods will not suffice for the standard-requirement linking task as proposed in this work, yielding to methods such as language embeddings or generative LLMs to perform such tasks which can incorporate context and topical relevance. This poses great potential to reduce workflows surrounding potentially safety-critical processes, such as the selection of standards to be applied to projects, often detailing safety guidelines for components or systems. In this light, firms and designers are recommended to ensure that large language models and such tools developed in this work still have human-in-the-loop verification procedures in place. The creation of new technology or tools that have arisen alongside the popularity of large language models hosts great potential to enhance an engineer's or firm's productivity. However, the liability remains placed on the engineer's shoulders for the advancement and assurance of the safety of society.

## 5.4 Limitations

The methods and results in this work are not without limitations. The various corpora used in the evaluation of the models and frameworks in this work are subject to inputs, or the richness of the training corpus or database contents, and would benefit from enhanced size and variance. While the researchers are confident that the conclusions drawn in this thesis are appropriate and sound, more data for the evaluation of this work would have proven useful. However, the researchers were subject to the feasible limits of paywalls enforced by many SDOs for their standard documents and also by the proprietary nature of engineering documentation surrounding product development. While the industry case study utilized for evaluation in this work is robust, the researchers realize that more validation cases would have led to greater confidence in the generalizability of the approaches. Additionally, the tasks in this work involve using standard tools in unconventional ways. As such, there are no established methods for evaluation within the domain. The limited availability of proprietary product documentation, which includes detailed product requirements and compliance standards, further introduces ambiguity into the evaluation methods.

# Chapter 6

# Conclusion and Future Work

This work establishes foundational knowledge on how large language models process two key components of engineering design documentation, design requirements and design standards. Through the training of a large language model, BERT, to distinguish between standards and requirements the researchers uncovered key linguistic differences present in standards and requirements that may be leveraged in future, optimized methods for these tasks surrounding design standards. This work has also presented the evaluation of three embedding-based techniques for retrieving project-relevant engineering standards. Our results demonstrate that ANN-based initial retrieval, followed by reranking using GPT Zero-shot, effectively and efficiently returns relevant engineering standards. The high precision of this approach enables comprehensive and time-saving search results for designers early in the design phases. Furthermore, the precision and additional relevancy filtering of this approach introduce less irrelevant noise into RAG systems, such as SARG, producing a more palatable integration of standards for designers. Moreover, the introduction of the SARG tool validates this framework as an effective means to further aid designers during the early conceptual phases by developing formal, high-level compliance requirements. These requirements aim to guide the design process and facilitate engineering decision-making procedures by bridging the gap in project-relevant standard retrieval.

# Appendix A

# UMAP Projections of SRC Manifold

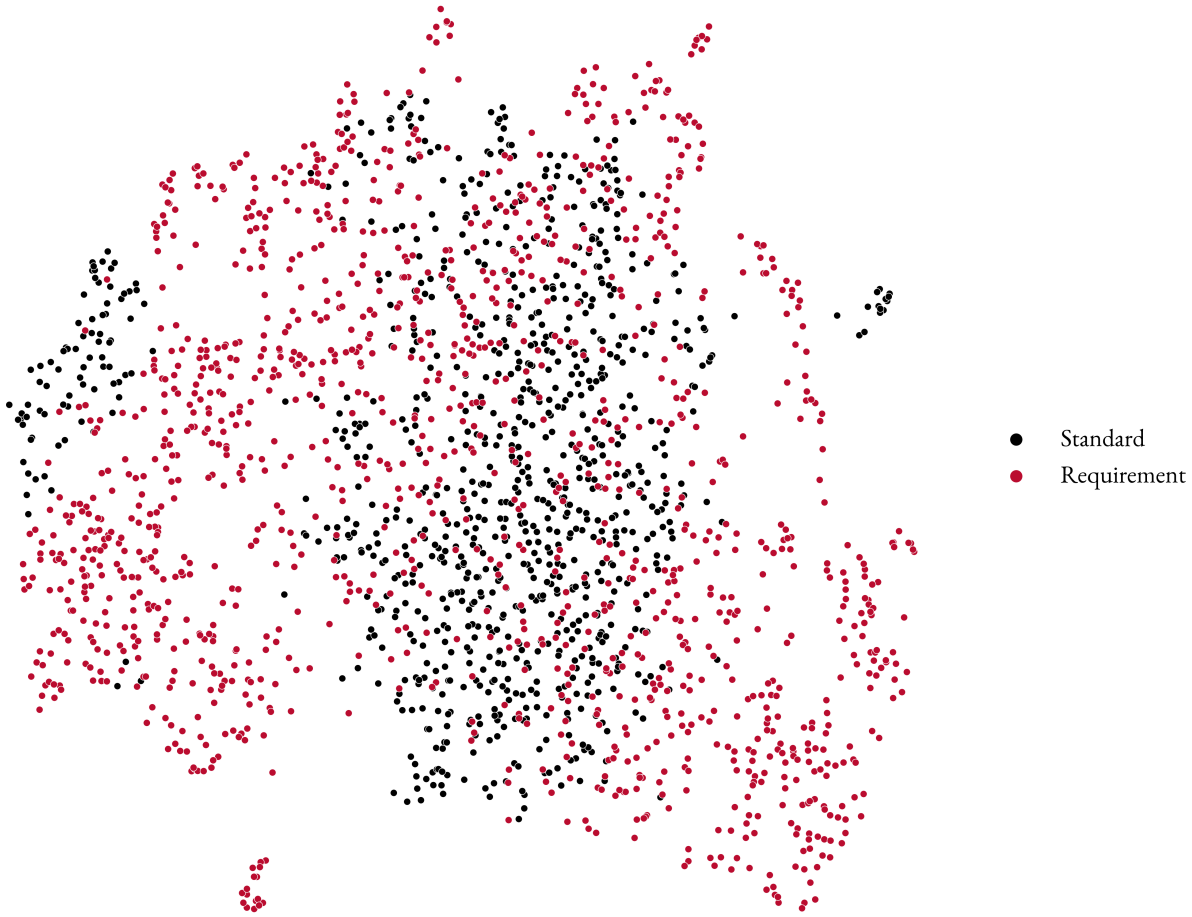


Figure A.1: UMAP Projection of Standards and Requirements by Enity


Figure A.2: UMAP Projection of Standards and Requirements by Enity

# Appendix B

# Term Frequency vs Word Attribution Plots

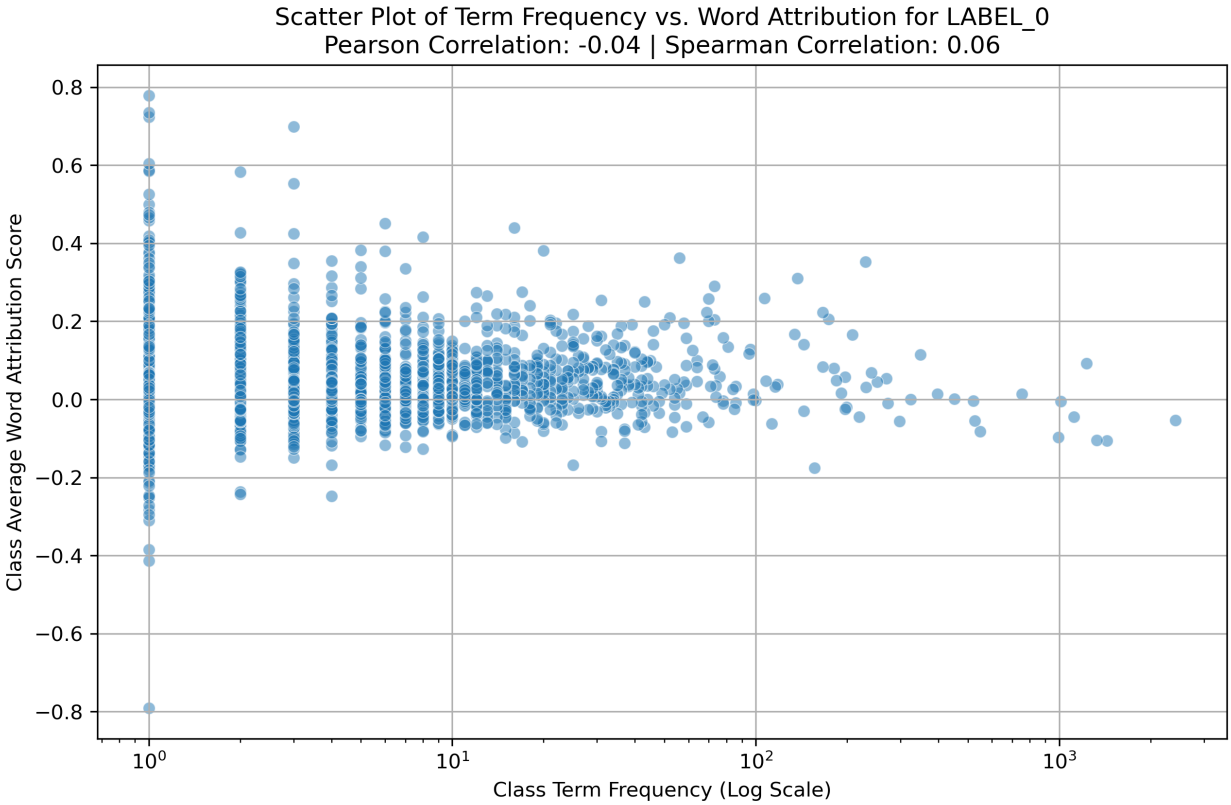


Figure B.1: Frequency vs Attribution for "Requirement" Class

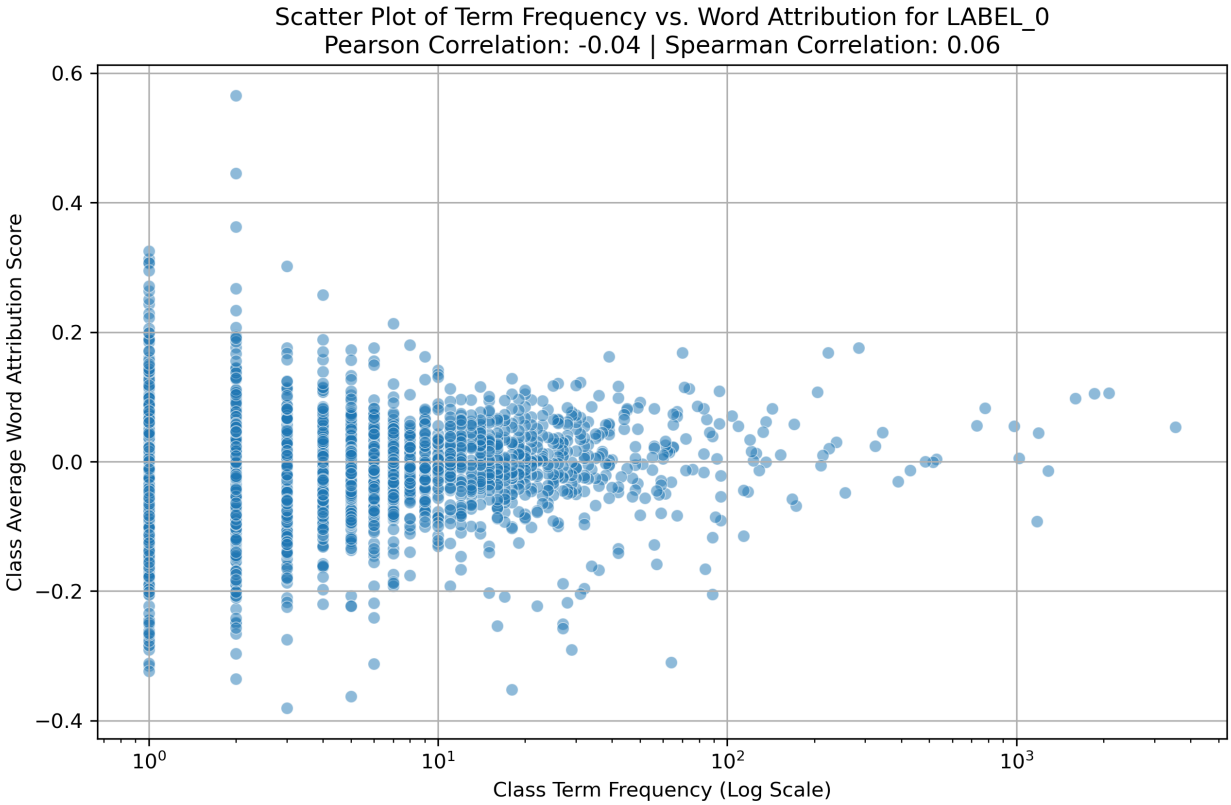


Figure B.2: Frequency vs Attribution for "Standard" Class

# Appendix C

# UMAP Embeddings of Search Results

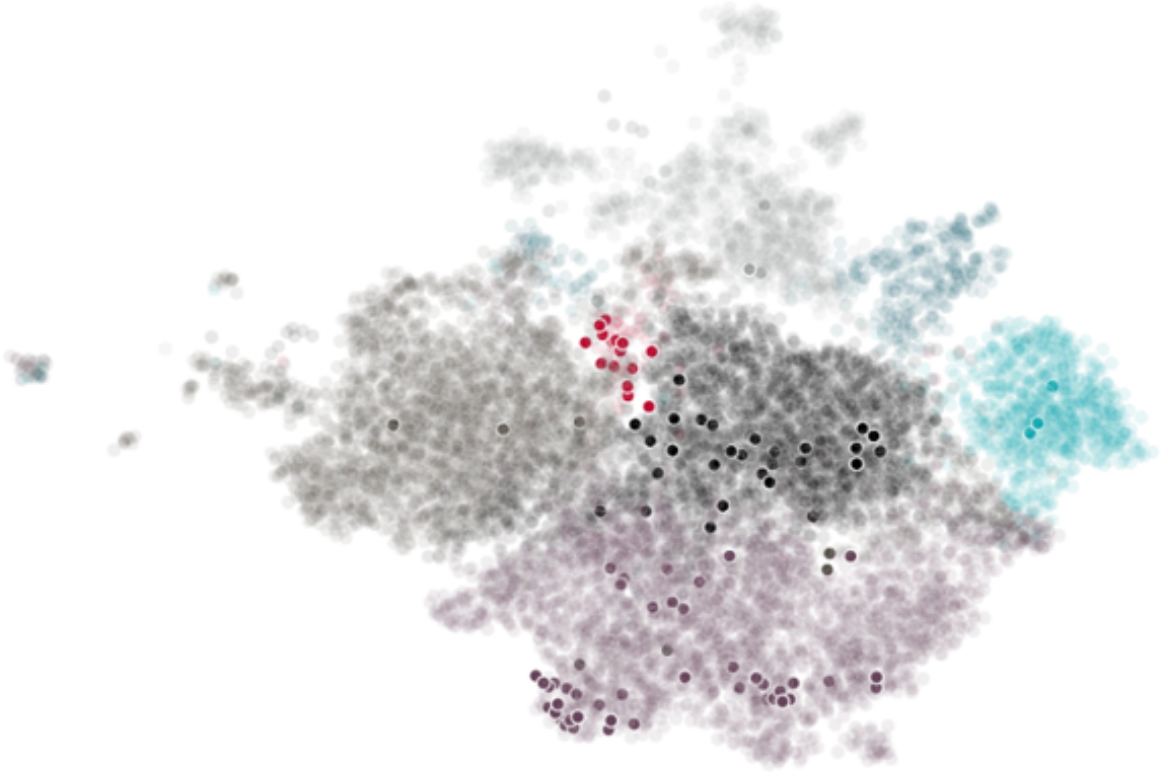

Figure C.1: UMAP Projection of ANNS Search Results on Latent Standard Manifold

Figure C.2: UMAP Projection of ANNS + Cohere Rerank Search Results on Latent Standard Manifold

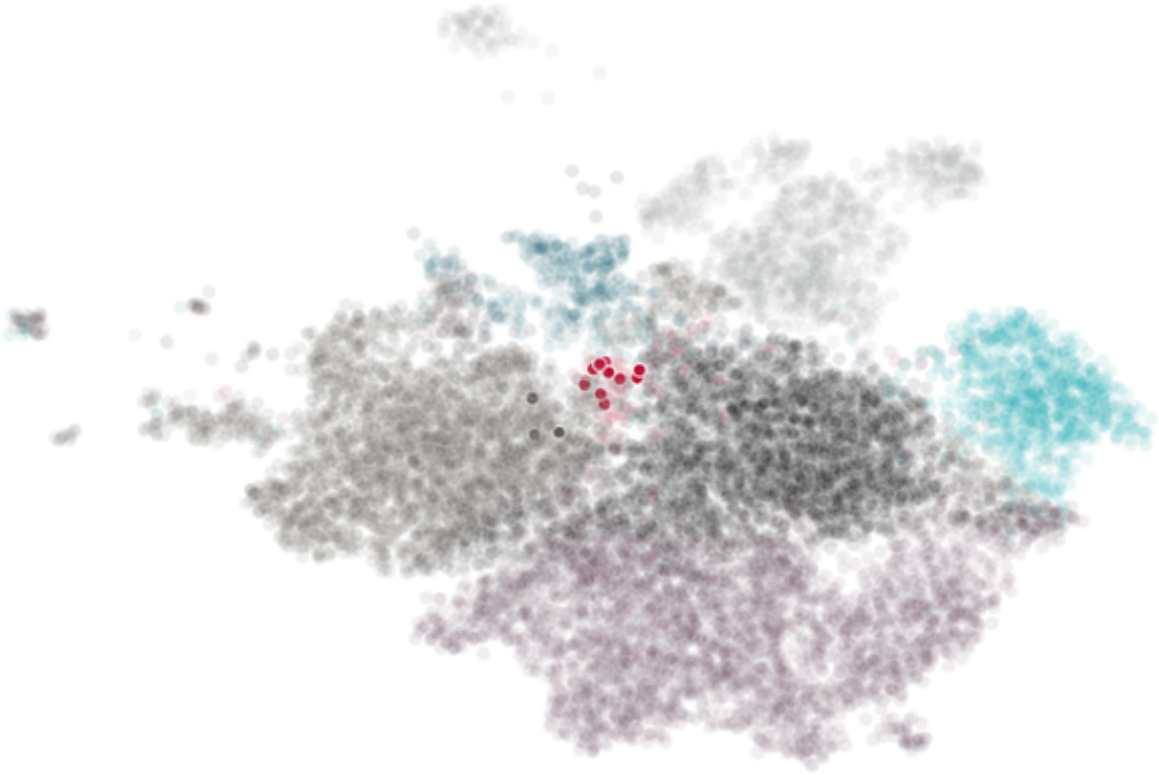

Figure C.3: UMAP Projection of ANNS + Zero-shot Rerank Search Results on Latent Standard Manifold

# Appendix D

# Code for Fine-tuning BERT on SRC Task

```
import numpy as np
import pandas as pd
from transformers import BertForSequenceClassification
from transformers import Trainer, TrainingArguments, EarlyStoppingCallback
from datasets import load_metric
from sklearn . metrics import matthews_corrcoef as mcc
import matplotlib.pyplot as plt
import seaborn as sns
from cf_matrix import make_confusion_matrix
```

```
df = pd.read_csv("STDREQLabeled.csv")
df = df.sample(frac=1, random_state=42).reset_index(drop=True)
df = df.loc[(df.label == "standard") | (df.label == "requirement")]
requirements = df.text.tolist()
labels = df.label.map({'standard': 1, 'requirement': 0}).tolist()
```

```
train_text, test_text, train_labels, test_labels = train_test_split(
    requirements,
    labels,
```

```
    random_state=41,
    test_size=0.2
)

train_text, val_text, train_labels, val_labels = train_test_split(
    train_text,
    train_labels,
    random_state=42,
    test_size=0.1
)
```

---

```
tokenizer = BertTokenizerFast.from_pretrained('bert-base-uncased')
train_encodings = tokenizer(train_text, truncation=True, padding=True)
val_encodings = tokenizer(val_text, truncation=True, padding=True)
test_encodings = tokenizer(test_text, truncation=True, padding=True)
```

---

```
model=BertForSequenceClassification.from_pretrained("bert-base-uncased")
metric = load_metric("matthews_correlation") #load MCC metric
def compute_metrics(eval_pred):
    predictions, labels = eval_pred
    predictions = np.argmax(predictions, axis=1)
    return metric.compute(predictions=predictions, references=labels)
```

---

```
class StdReqDataset(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 = StdReqDataset(train_encodings, train_labels)
val_dataset = StdReqDataset(val_encodings, val_labels)
test_dataset = StdReqDataset(test_encodings, test_labels)
```

---

```
training_args = TrainingArguments(
    output_dir='./results',
    evaluation_strategy='epoch',
    save_strategy='epoch',
    num_train_epochs=6,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    learning_rate=2e-5,
    warmup_ratio=0.1,
    weight_decay=0.01,
    logging_dir='./logs',
    logging_strategy='epoch',
    load_best_model_at_end=True,
    metric_for_best_model='eval_loss',
)
```

---

```
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
    compute_metrics=compute_metrics,
    callbacks=[
        EarlyStoppingCallback(early_stopping_patience=1,
   early_stopping_threshold=0.01)
```

```
    ]
)
```

---

```
trainer.train()
```

---

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

---

```
sns.set_context('paper')
labels = ['TN','FP','FN','TP']
categories = ['Requirement', 'Standard']
cmap = sns.color_palette('rocket', n_colors=200)
make_confusion_matrix(cf_matrix, categories=categories, cmap=cmap)
plt.savefig('confusionmatrixsrc.png', dpi=300)
```

---

```
model.save_pretrained('savedmodels')
model.config.save_pretrained('savedmodels')
tokenizer.save_pretrained('savedmodels')
```

---

# Appendix E

# Analysing Fine-tuned Model

## E.1 Executing Integrated Gradients

```
from transformers import AutoModelForSequenceClassification, AutoTokenizer
import pandas as pd
import json
from collections import defaultdict
from transformers_interpret import SequenceClassificationExplainer
from sklearn.model_selection import train_test_split
```

```
df = pd.read_csv("STDREQLabeled.csv")
df = df.sample(frac=1, random_state=42).reset_index(drop=True)
df = df.loc[(df.label == "standard") | (df.label == "requirement")]
requirements = df['text'].tolist()
labels = df.label.map({'standard': "LABEL_1", 'requirement': "LABEL_0"}).
    tolist()
df_new = pd.DataFrame({'text': requirements, 'label': labels})
```

```
model_directory = "savedmodels/"
model = AutoModelForSequenceClassification.from_pretrained(model_directory
    )
```

```
tokenizer = AutoTokenizer.from_pretrained(model_directory)
cls_explainer = SequenceClassificationExplainer(model, tokenizer)
```

```
word_attributions_aggregated = {
    "LABEL_0": defaultdict(list),  # For 'requirement'
    "LABEL_1": defaultdict(list)   # For 'standard'
}
```

```
for index, row in df_new.iterrows():
    text = row['text']
    # For each class, calculate and collect word attributions
    for class_name in ["LABEL_0", "LABEL_1"]:
        word_attributions = cls_explainer(text, class_name=class_name)
        # Aggregate attributions by word for the current class
        for word, score in word_attributions:
            word_attributions_aggregated[class_name][word].append(score)
```

```
word_attributions_avg = {
    class_name: {word: sum(scores) / len(scores) for word, scores in
   class_attributions.items()}
    for class_name, class_attributions in word_attributions_aggregated.
   items()
}
```

```
difference_in_attributions = {
    word: word_attributions_avg["LABEL_1"].get(word, 0) -
   word_attributions_avg["LABEL_0"].get(word, 0)
    for word in set(word_attributions_avg["LABEL_1"]) | set(
   word_attributions_avg["LABEL_0"])
}
```

```
with open('word_attributions_avg.json', 'w') as f:
    json.dump(word_attributions_avg, f)
```

```
with open('difference_in_attributions.json', 'w') as f:
    json.dump(difference_in_attributions, f)
```

## E.2 Creating Word Attribution Histogram

```
import json
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib import rcParams
```

```
file_path = "difference_in_attributions.json"
with open(file_path, 'r') as file:
    attributions_data = json.load(file)
```

```
attribution_values = np.array(list(attributions_data.values()))
```

```
sns.set_theme(style="whitegrid")
rcParams.update({'font.size': 14})
plt.figure(figsize=(12, 6))
sns.histplot(attribution_values, bins=50, kde=False, color="skyblue",
    edgecolor="black", label='Attribution Values', log_scale=(False, True))
plt.xlabel('Attribution Value', fontsize=16)
plt.ylabel('Frequency', fontsize=16)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.axvline(np.mean(attribution_values), color='red', linestyle='dashed',
    linewidth=1, label='Mean Attribution')
plt.legend(fontsize=13, title_fontsize=14)
plt.grid(True, which='both', linestyle='--', linewidth=0.5)
plt.savefig('attribution_values_distribution.png', dpi=300, format='png',
    bbox_inches='tight')
```

## E.3 Creating Word Clouds

```
import json
import numpy as np
from wordcloud import WordCloud
import matplotlib.pyplot as plt
```

```
file_path = "difference_in_attributions.json"
with open(file_path, 'r') as file:
    attributions_data = json.load(file)
```

```
attribution_values = np.array(list(attributions_data.values()))
```

```
lower_bound = np.percentile(attribution_values, 37.5)
upper_bound = np.percentile(attribution_values, 62.5)
```

```
filtered_attributions = {
    token: value for token, value in attributions_data.items()
    if (lower_bound < value < upper_bound) and not token.isdigit() and '#'
    not in token
}
```

```
positive_attributions = {token: value for token, value in
   filtered_attributions.items() if value > 0}
negative_attributions = {token: abs(value) for token, value in
   filtered_attributions.items() if value < 0}
```

```
wordcloud_pos = WordCloud(width=800, height=400, background_color=None,
   mode='RGBA').generate_from_frequencies(positive_attributions)
wordcloud_neg = WordCloud(width=800, height=400, background_color=None,
   mode='RGBA').generate_from_frequencies(negative_attributions)
```

```
def save_wordcloud(wordcloud, filename, dpi=300):
    plt.figure(figsize=(10, 5))
```

```
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.axis('off')
    plt.savefig(filename, format='png', transparent=True, dpi=dpi)

save_wordcloud(wordcloud_pos, 'standard_attributions.png')
save_wordcloud(wordcloud_neg, 'requirement_attributions.png')
```

# Appendix F

# Code for SRC Task

## F.1 Sampling Random Project Requirements

```
import pandas as pd
```

```
file_path = "Sentence Requirements with document labels.xlsx"
df = pd.read_excel(file_path)
```

```
label_column = 'document'
unique_labels = df[label_column].unique()
print(f"Unique Labels: {unique_labels}")
```

```
samples_per_label = 5
sampled_dfs = []
```

```
for label in unique_labels:
    df_label = df[df[label_column] == label]
    sampled_df = df_label.sample(n=samples_per_label, replace=False,
   random_state=2)
    sampled_dfs.append(sampled_df)
```

```
final_sampled_df = pd.concat(sampled_dfs, ignore_index=True)
output_path = "Sampled Sentence Requirements with document labels.xlsx"
```

```
final_sampled_df.to_excel(output_path, index=False)
```

## F.2 Upserting Standard Sub-clauses to Pinecone

```
import openai
from openai import OpenAI
import pinecone
import pandas as pd
import numpy as np
```

```
client = OpenAI(api_key= '1234')

pinecone.init(
  api_key='1234',
  environment='gcp-starter'
)
index = pinecone.Index('standard-vectorstore')
```

```
csv_file_path = 'standard_labeled.csv'
df = pd.read_csv(csv_file_path)
print(df.head())
```

```
def get_embedding(text, model="text-embedding-ada-002"):
    embedding = client.embeddings.create(input=[text], model=model).data
    [0].embedding
    return embedding

def get_embeddings_batch(texts):
    embeddings = []
    for text in texts:
        embeddings.append(get_embedding(text))
    return embeddings
```

```
df['batch_id'] = np.arange(len(df)) // 20

unique_batches = df['batch_id'].unique()
for batch_id in unique_batches:
    batch_df = df[df['batch_id'] == batch_id]
    batch_texts = batch_df['text'].tolist()
    batch_embeddings = get_embeddings_batch(batch_texts)
    data_to_upsert = [(str(id), embedding) for id, embedding in zip(
   batch_df.index, batch_embeddings)]

    index.upsert(vectors=data_to_upsert, namespace="SentencesNewFiltered")

    print(f"Processed and upserted batch {batch_id}")

print("All batches processed.")
```

## F.3 Executing SRC Task

```
import pinecone
import numpy as np
import openai
from openai import OpenAI
import tiktoken
import csv
import time
import pandas as pd
```

```
pinecone.init(
  api_key='1234',
  environment='gcp-starter'
)
```

```
pindex = pinecone.Index('standard-vectorstore')
client = OpenAI(api_key= '1234')
```

---

```
csv_file_path = 'standard_labeled.csv'
df_standards = pd.read_csv(csv_file_path, usecols=['text', 'document'])
df_standards['ID'] = df_standards.index
print(df_standards.head())
```

---

```
csv_file_path = 'sampledrequirementsforEVAL.csv'
df_requirements = pd.read_csv(csv_file_path)
df_requirements['ID'] = df_requirements.index + 1
print(df_requirements.head())
df_requirements['document'] = df_requirements['document'].astype(str)
```

---

```
def get_embedding(text, model="text-embedding-ada-002", max_tokens=8191):
    encoding = tiktoken.encoding_for_model("text-embedding-ada-002")
    tokens = encoding.encode(text)
if len(tokens) > max_tokens:
        tokens = tokens[:max_tokens]
    truncated_content = encoding.decode(tokens)
return client.embeddings.create(input=[truncated_content],
       model=model).data[0].embedding
```

---

```
document_ids_per_input = {}

for idx, row in df_inputrequirements.iterrows():
    try:
        requirement_embedding = get_embedding(row['text'])
        pinecone_results = pindex.query(vector=requirement_embedding,
   top_k=3, namespace='SentencesNewFiltered', include_values=False)
        document_ids = [result['id'] for result in pinecone_results['
   matches']]
        document_ids_per_input[idx] = document_ids
```

```
        print(f"Processed row {idx}, retrieved {len(document_ids)}
    document IDs")
    except Exception as e:
        print(f"Error processing row {idx}: {e}")
```

```
structured_results = []
df_standards['ID'] = df_standards['ID'].astype(str)

for input_idx, document_ids in document_ids_per_input.items():
    filtered_docs = df_standards[df_standards['ID'].isin(document_ids)]
    for _, doc in filtered_docs.iterrows():
        structured_results.append({
            "Input Index": input_idx,
            "Document Text": doc['text']
        })

df_results = pd.DataFrame(structured_results)
csv_file_name = 'sentenceresultsforeval.csv'
df_results.to_csv(csv_file_name, index=False)
```

# Appendix G

# Code for Standard Search Framework

## G.1 Upserting Standard Document Chunks to Pinecone

```
import os
import openai
from openai import OpenAI
import pandas as pd
import time
import pickle
import pinecone
import tiktoken
```

```
pinecone.init(
  api_key='1234',
  environment='gcp-starter'
)
index = pinecone.Index('standard-vectorstore')
client = OpenAI(api_key= '1234')
```

```
def chunk_tokens(file_content: str, file_name: str, token_limit: int =
    1000):
    if not os.path.exists('chunks'):
        os.makedirs('chunks')

    enc = tiktoken.encoding_for_model("text-embedding-ada-002")
    chunks = []
    tokens = enc.encode(file_content, disallowed_special=())

    chunk_number = 1

    while tokens:
        chunk = tokens[:token_limit]
        chunk_text = enc.decode(chunk)
        last_punctuation = max(chunk_text.rfind("."), chunk_text.rfind("?"
    ),
                               chunk_text.rfind("!"), chunk_text.rfind("\n
    "))
        if last_punctuation != -1 and len(tokens) > token_limit:
            chunk_text = chunk_text[:last_punctuation + 1]
        cleaned_text = chunk_text.replace("\n", " ").strip()
        if cleaned_text and not cleaned_text.isspace():
            chunks.append(cleaned_text)
            chunk_file_name = f"{file_name}_{chunk_number}.txt"
            with open(os.path.join('chunks', chunk_file_name), "w") as
    file:
                file.write(cleaned_text)
            chunk_number += 1
        tokens = tokens[len(enc.encode(chunk_text, disallowed_special=()))
    :]
    return chunks
```

```
def read_text_file(file_path):
    with open(file_path, 'r', encoding='utf-8') as file:
        return file.read()
```

---

```
folder_path = 'documenttextfiles/'
for file_name in os.listdir(folder_path):
    if file_name.endswith('.txt'):
        base_file_name = os.path.splitext(file_name)[0]
        file_path = os.path.join(folder_path, file_name)
        file_content = read_text_file(file_path)
        chunks = chunk_tokens(file_content, base_file_name)
```

---

```
def get_embedding(file_content, model="text-embedding-ada-002", max_tokens
   =8191):
    encoding = tiktoken.encoding_for_model("text-embedding-ada-002")
    tokens = encoding.encode(file_content)
    return client.embeddings.create(input=[file_content], model=model).
   data[0].embedding
```

---

```
chunk_folder = 'chunks/'
file_count = 0
pinecone_data = []
for file_name in os.listdir(chunk_folder):
    if file_name.endswith('.txt'):
        file_path = os.path.join(chunk_folder, file_name)
        file_content = read_text_file(file_path)
        embedding = get_embedding(file_content)
        pinecone_data.append((file_name, embedding))

        file_count += 1
        print(f"Embedded files: {file_count}")
```

---

```
with open('validation_embeddings_chunks.pkl', 'wb') as f:
    pickle.dump(pinecone_data, f)
```

```
print("Embeddings saved for Pinecone.")
```

```
batch_size = 100
for i in range(0, len(pinecone_data), batch_size):
    batch = pinecone_data[i:i + batch_size]
    index.upsert(vectors=batch, namespace="Chunks")
```

## G.2 ANNS Search

```
import pinecone
import numpy as np
import openai
from openai import OpenAI
import tiktoken
import pandas as pd
import csv
```

```
pinecone.init(
  api_key='1234',
  environment='gcp-starter'
)
index = pinecone.Index('standard-vectorstore')
client = OpenAI(api_key= '1234')
```

```
def get_embedding(file_content, model="text-embedding-ada-002", max_tokens
    =8191):
    encoding = tiktoken.encoding_for_model("text-embedding-ada-002")
    tokens = encoding.encode(file_content)
    return client.embeddings.create(input=[file_content], model=model).
    data[0].embedding
```

```
document_ids = []
try:

    with open('IndustryProductDescription.txt', 'r', encoding='utf-8') as
   file:
        file_content = file.read()

    embedding = get_embedding(file_content)

    pinecone_results = index.query(vector=embedding, top_k=100, namespace=
   'Chunks', include_values=False)

    document_ids = [result['id'] for result in pinecone_results['matches'
   ]]
    print(f"Retrieved {len(document_ids)} document IDs")

except Exception as e:
    print(f"Error processing the text file: {e}")
```

---

```
table = [("ID", "Score")]
for match in pinecone_results["matches"]:
    table.append((match["id"], match["score"]))
```

---

```
folder_path = '1000 Token Chunked Pinecone Artifacts/'
files_content_dict = {}
filenames_ordered = []
```

---

```
files_content_dict = {}
filenames_ordered = []

for row in table[1:]:
    file_name = row[0]
    file_path = os.path.join(folder_path, file_name)
```

```
    if os.path.exists(file_path):
        with open(file_path, 'r', encoding='utf-8') as file:
            content = file.read()
            files_content_dict[file_name] = content
            filenames_ordered.append(file_name)
```

```
df = pd.DataFrame({
    "Filename": filenames_ordered
})

print(df)

excel_file_path = 'ANNSResults.xlsx'
df.to_excel(excel_file_path, index=False)
print(f"DataFrame saved to {excel_file_path}")
```

## G.3 ANNS Search with Cohere Rerank

```
import pinecone
import numpy as np
import openai
from openai import OpenAI
import tiktoken
import pandas as pd
import csv
```

```
pinecone.init(
  api_key='1234',
  environment='gcp-starter'
)
index = pinecone.Index('standard-vectorstore')
```

```
client = OpenAI(api_key= '1234')
```

```
def get_embedding(file_content, model="text-embedding-ada-002", max_tokens
    =8191):
     encoding = tiktoken.encoding_for_model("text-embedding-ada-002")
     tokens = encoding.encode(file_content)
     return client.embeddings.create(input=[file_content], model=model).
    data[0].embedding
```

```
document_ids = []
try:

     with open('IndustryProductDescription.txt', 'r', encoding='utf-8') as
    file:
          file_content = file.read()

     embedding = get_embedding(file_content)

     pinecone_results = index.query(vector=embedding, top_k=100, namespace=
    'Chunks', include_values=False)

     document_ids = [result['id'] for result in pinecone_results['matches'
    ]]
     print(f"Retrieved {len(document_ids)} document IDs")

except Exception as e:
     print(f"Error processing the text file: {e}")
```

```
table = [("ID", "Score")]
for match in pinecone_results["matches"]:
     table.append((match["id"], match["score"]))

folder_path = '1000 Token Chunked Pinecone Artifacts/'
```

```
files_content_dict = {}
filenames_ordered = []
for row in table[1:]:
    file_name = row[0]
    file_path = os.path.join(folder_path, file_name)

    if os.path.exists(file_path):
        with open(file_path, 'r', encoding='utf-8') as file:
            content = file.read()
            files_content_dict[file_name] = content
            filenames_ordered.append(file_name)
```

---

```
contents_for_reranking = list(files_content_dict.values())

rerank_results = co.rerank(
    query=file_content,
    documents=contents_for_reranking,
    top_n=100,
    model="rerank-english-v2.0",
)
reranked_filenames = [filenames_ordered[result.index] for result in
   rerank_results]
```

---

```
structured_results = []
for result in rerank_results:
    structured_results.append({
        "Document Index": reranked_filenames[result.index],
        "Relevance Score": result.relevance_score
    })

df_results = pd.DataFrame(structured_results)
```

---

```
excel_file_path = 'ANNSCohereResults.xlsx'
dfresults.to_excel(excel_file_path, index=False)
print(f"DataFrame saved to {excel_file_path}")
```

## G.4 ANNS Search with GPT Rerank

```
import pinecone
import numpy as np
import openai
from openai import OpenAI
import tiktoken
import csv
import time
import pandas as pd
from tenacity import retry, wait_random_exponential, stop_after_attempt
import math
import os
```

```
pinecone.init(
  api_key='1234',
  environment='gcp-starter'
)
pindex = pinecone.Index('standard-vectorstore')
client = OpenAI(api_key= '1234')
OPENAI_MODEL = "gpt-4-turbo-preview"
```

```
def get_embedding(file_content, model="text-embedding-ada-002", max_tokens
    =8191):
     encoding = tiktoken.encoding_for_model("text-embedding-ada-002")
     tokens = encoding.encode(file_content)
     return client.embeddings.create(input=[file_content], model=model).
    data[0].embedding
```

```
document_ids = []
reranked_docs = []
query_filename = 'IndustryProductDescription.txt'

try:
    with open(query_filename, 'r', encoding='utf-8') as file:
        file_content = file.read()

    requirement_embedding = get_embedding(file_content)
    pinecone_results = pindex.query(vector=requirement_embedding, top_k=
   num_to_return, namespace='Chunks', include_values=False)

    document_ids = [result['id'] for result in pinecone_results['matches'
   ]]
    print(f"Retrieved {len(document_ids)} document IDs")
    df_export = pd.DataFrame(document_ids, columns=['ID'])

    csv_filename = 'pinecone_topk.csv'
    df_export.to_csv(csv_filename, index=False)

except Exception as e:
    print(f"Error processing the text file: {e}")
```

```
table = [("ID", "Score")]
for match in pinecone_results["matches"]:
    table.append((match["id"], match["score"]))

folder_path = '1000 Token Chunked Pinecone Artifacts/'

files_content_dict = {}
filenames_ordered = []
```

```
for row in table[1:]:
    file_name = row[0]
    file_path = os.path.join(folder_path, file_name)

    if os.path.exists(file_path):
        with open(file_path, 'r', encoding='utf-8') as file:
            content = file.read()
            files_content_dict[file_name] = content
            filenames_ordered.append(file_name)

documentsforreranking = list(files_content_dict.values())
```

---

```
tokens = [" Yes", " No"]
tokenizer = tiktoken.encoding_for_model(OPENAI_MODEL)
ids = [tokenizer.encode(token) for token in tokens]
ids[0], ids[1]
#7566, 2360
```

---

```
CLASSIFICATION_PROMPT = """You are an Assistant responsible for helping
   detect whether the retrieved engineering standard is relevant to the
   project description. For a given input, you need to output a single
   token: "Yes" or "No" indicating the retrieved standard is relevant to
   the project description query.
Project Description: {query}
Engineering Standard: {document}"""
```

---

```
@retry(wait=wait_random_exponential(min=1, max=40), stop=
   stop_after_attempt(3))
def document_relevance(query, document):
    response = client.chat.completions.create(
        model=OPENAI_MODEL,
        messages=[{
```

```
            "role":
            "user",
            "content":
            CLASSIFICATION_PROMPT.format(query=query, document=document)
        }],
        logprobs=True,
        top_logprobs=1,
        temperature=0,
        logit_bias={
            7566: 1,
            2360: 1
        },
        max_tokens=2,
    )

    return response
```

---

```
query = file_content
output_list = []
file_output_list = []
probability_list = []

for file_name, content in files_content_dict.items():
    try:
        output = document_relevance(query, document=content)
        output_list.append(output.choices[0].message.content)
        file_output_list.append(file_name)
        logprob = output.choices[0].logprobs.content[0].logprob
        probability_list.append(math.exp(logprob))
    except Exception as e:
        print(f"Error processing document {file_name}: {e}")
```

---

```
output_df = pd.DataFrame({
```

```
    "Relevant?": output_list,
    "Filename": file_output_list,
    "Relevancy Percentage":probability_list
})
print(output_df)
```

---

```
output_df['Sort_Yes'] = (output_df['Relevant?'] == 'Yes').astype(int)
reranked_df = output_df.sort_values(by=['Sort_Yes', 'Relevancy Percentage'
    ], ascending=[False, False])
reranked_df = reranked_df.drop(columns=['Sort_Yes'])
reranked_df = reranked_df.reset_index(drop=True)
reranked_rel_df = reranked_df[reranked_df['Relevant?'] == 'Yes']
reranked_rel_df = reranked_rel_df.reset_index(drop=True)
num_rows = len(reranked_rel_df)
print(f"The number of relevant standards returned for your project is {
    num_rows}")
```

---

```
excel_file_path = "returnedstandardsgptreranked.xlsx"
reranked_rel_df.to_excel(excel_file_path, index=True)
print(f"DataFrame saved to {excel_file_path}")
```

---

## G.5 Plotting Search Framework Precision

---

```
from pathlib import Path
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
```

---

```
path1 = Path("Validation Case Retrieval/resultswithGPTreranker.xlsx")
df1 = pd.read_excel(path1)
precision_values1 = df1['Precision'].head(100)
```

```
precision_percentages1 = precision_values1 * 100
df_percentages = pd.DataFrame({'ANNS + GPT ZERO Shot':
    precision_percentages1.values})
print(df_percentages.head())
```

---

```
path2 = Path("Validation Case Retrieval/KMCresultswithcoherereranker.xlsx"
    )
df2 = pd.read_excel(path2)
df_percentages['ANNS + Cohere'] = df2['Precision'].head(100) * 100
print(df_percentages.head())
```

---

```
path3 = Path("Validation Case Retrieval/VAL_Case_top_100.csv")
df3 = pd.read_csv(path3)
df_percentages['ANNS'] = df3['Precision'].head(100) * 100
print(df_percentages.head(30))
```

---

```
df_reset = df_percentages.reset_index()
df_melted = df_reset.melt(id_vars=["index"], var_name="Model", value_name=
    "Precision")

models_to_plot = ['ANNS', 'ANNS + Cohere', 'ANNS + GPT ZERO Shot']
df_melted = df_melted[df_melted['Model'].isin(models_to_plot)]

plt.figure(figsize=(12, 6))
sns.lineplot(data=df_melted, x="index", y="Precision", hue="Model", style=
    "Model", markers=False, dashes=True)

plt.xlabel('Index (top K)', fontsize=16)
plt.ylabel('Precision (%)', fontsize=16)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.legend(title='Search Method', fontsize=13, title_fontsize=14)
plt.gca().xaxis.set_major_locator(ticker.MaxNLocator(nbins=22))
```

```
plt.grid(True)
plt.tight_layout()
plt.xlim(0, 101)

outputpath = Path("Validation Case Retrieval/cosine_cohere_GPT_all")
plt.savefig(outputpath, dpi=300)
plt.show()
```

## G.6 Visualizing Search Output Manifold Location

```
import pinecone
import pickle
import numpy as np
import pandas as pd
import os
import umap
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.font_manager as font_manager
```

```
pinecone.init(
  api_key='1234',
  environment='gcp-starter'
)
```

```
def get_filenames(folder_path):
    filenames = []
    for filename in os.listdir(folder_path):
        if os.path.isfile(os.path.join(folder_path, filename)):
            filenames.append(filename)
    return filenames
```

```
folder_path = '1000 Token Chunked Pinecone Artifacts/'
file_list = get_filenames(folder_path)
df_ids = pd.DataFrame(file_list, columns=['ID'])
```

---

```
with open('1000 Token Chunked Pinecone Artifacts/
   pinecone_embeddings_chunks.pkl', 'rb') as f:
    pinecone_data_1 = pickle.load(f)

with open('1000 Token Chunked Pinecone Artifacts/
   validation_embeddings_chunks.pkl', 'rb') as f:
    pinecone_data_2 = pickle.load(f)

df1 = pd.DataFrame(pinecone_data_1, columns=['filename', 'embedding'])
df2 = pd.DataFrame(pinecone_data_2, columns=['filename', 'embedding'])
df = pd.concat([df1, df2], ignore_index=True)
print(df.head())
```

---

```
embedding_df = pd.DataFrame(df['embedding'].tolist())
result_df = pd.concat([df['filename'], embedding_df], axis=1)
df = result_df
print(df.head())
```

---

```
top_100_filenames_df = pd.read_csv('GPTreranked.csv')
top_100_filenames = set(top_100_filenames_df['ID'])

folders_to_scan = [
    'Document Level Text Files/AASHTO Text Files/',
    'Document Level Text Files/AATCC Text Files/',
    'Document Level Text Files/AHRI Text Files/',
    'Document Level Text Files/ASTM Text Files/',
    'Document Level Text Files/AWWA Text Files/',
    'Document Level Text Files/CCSDS TEXT Files/',
    'Document Level Text Files/IEEE Text Files/',
```

```
    'Document Level Text Files/ASABE Text Files/',
]

def find_matching_folder(file_id, folders):
    for folder in folders:
        if os.path.exists(folder):
            for file in os.listdir(folder):
                file_prefix = file[:5]
                if file_prefix in file_id:
                    return folder
    return None

def is_top_100(file_id, top_100_filenames):
    return file_id in top_100_filenames

df['Matching_Folder'] = df['filename'].apply(lambda x:
   find_matching_folder(x, folders_to_scan))
df['Is_Top_100'] = df['filename'].apply(lambda x: is_top_100(x,
   top_100_filenames))

df.sort_values('Matching_Folder', inplace=True)
folder_counts = df['Matching_Folder'].value_counts()

print(df)
print(folder_counts

features = df.drop(columns=['filename', 'Matching_Folder', 'Is_Top_100'])
```

---

```
umap_model = umap.UMAP(n_neighbors=250, min_dist=1, n_components=2,
   random_state=42, metric='cosine')
umap_projection = umap_model.fit_transform(features)
```

```
font_path = 'eb-garamond.12-regular.ttf'
eb_garamond = font_manager.FontProperties(fname=font_path, size=22)

hex_colors = ['#000000', '#00A3AD', '#004E60', '#BA0C2F', '#554F47',
              '#66435A', '#9EA2A2', '#D6D2C4', '#C8D8EB']
custom_palette = sns.color_palette(hex_colors)

plt.figure(figsize=(12, 8))
ax = plt.gca()

categories = df['Matching_Folder'].unique()
category_palette = {category: color for category, color in zip(categories,
    custom_palette)}

for category in categories:
    idx = df['Matching_Folder'] == category
    alpha_values = df.loc[idx, 'Is_Top_100'].map({True: 1.0, False: 0.09})
    alpha_values = alpha_values.clip(0, 1)
    ax.scatter(
        umap_projection[idx, 0],
        umap_projection[idx, 1],
        color=category_palette[category],
        alpha=alpha_values,
        label=category,
        s=50,
        edgecolors='white',
        linewidth=1
    )

plt.savefig('umap_projectionsearch.png', dpi=300, bbox_inches='tight',
   format='png')
plt.show()
```

# Appendix H

# Code for SARG

```
import pinecone
import numpy as np
import openai
from openai import OpenAI
import tiktoken
import csv
import time
import pandas as pd
from tenacity import retry, wait_random_exponential, stop_after_attempt
import math
import os
```

```
pinecone.init(
  api_key='1234',
  environment='gcp-starter'
)
pindex = pinecone.Index('standard-vectorstore')
client = OpenAI(api_key= '1234')
OPENAI_MODEL = "gpt-4-turbo-preview"
```

```
def get_embedding(file_content, model="text-embedding-ada-002", max_tokens
    =8191):
    encoding = tiktoken.encoding_for_model("text-embedding-ada-002")
    tokens = encoding.encode(file_content)
    return client.embeddings.create(input=[file_content], model=model).
    data[0].embedding
```

---

```
document_ids = []
reranked_docs = []
query_filename = 'IndustryProductDescription.txt'

try:
    with open(query_filename, 'r', encoding='utf-8') as file:
        file_content = file.read()

    requirement_embedding = get_embedding(file_content)
    pinecone_results = pindex.query(vector=requirement_embedding, top_k=
    num_to_return, namespace='Chunks', include_values=False)

    document_ids = [result['id'] for result in pinecone_results['matches'
    ]]
    print(f"Retrieved {len(document_ids)} document IDs")
    df_export = pd.DataFrame(document_ids, columns=['ID'])

    csv_filename = 'pinecone_topk.csv'
    df_export.to_csv(csv_filename, index=False)

except Exception as e:
    print(f"Error processing the text file: {e}")
```

---

```
table = [("ID", "Score")]
for match in pinecone_results["matches"]:
    table.append((match["id"], match["score"]))
```

```
folder_path = '1000 Token Chunked Pinecone Artifacts/'

files_content_dict = {}
filenames_ordered = []

for row in table[1:]:
    file_name = row[0]
    file_path = os.path.join(folder_path, file_name)

    if os.path.exists(file_path):
        with open(file_path, 'r', encoding='utf-8') as file:
            content = file.read()
            files_content_dict[file_name] = content
            filenames_ordered.append(file_name)

documentsforreranking = list(files_content_dict.values())
```

---

```
tokens = [" Yes", " No"]
tokenizer = tiktoken.encoding_for_model(OPENAI_MODEL)
ids = [tokenizer.encode(token) for token in tokens]
ids[0], ids[1]
#7566, 2360
```

---

```
CLASSIFICATION_PROMPT = """You are an Assistant responsible for helping
   detect whether the retrieved engineering standard is relevant to the
   project description. For a given input, you need to output a single
   token: "Yes" or "No" indicating the retrieved standard is relevant to
   the project description query.
Project Description: {query}
Engineering Standard: {document}"""
```

---

```
@retry(wait=wait_random_exponential(min=1, max=40), stop=
    stop_after_attempt(3))
def document_relevance(query, document):
    response = client.chat.completions.create(
        model=OPENAI_MODEL,
        messages=[{
            "role":
            "user",
            "content":
            CLASSIFICATION_PROMPT.format(query=query, document=document)
        }],
        logprobs=True,
        top_logprobs=1,
        temperature=0,
        logit_bias={
            7566: 1,
            2360: 1
        },
        max_tokens=2,
    )

    return response
```

---

```
query = file_content
output_list = []
file_output_list = []
probability_list = []

for file_name, content in files_content_dict.items():
    try:
        output = document_relevance(query, document=content)
        output_list.append(output.choices[0].message.content)
```

```
        file_output_list.append(file_name)
        logprob = output.choices[0].logprobs.content[0].logprob
        probability_list.append(math.exp(logprob))
    except Exception as e:
        print(f"Error processing document {file_name}: {e}")
```

---

```
output_df = pd.DataFrame({
    "Relevant?": output_list,
    "Filename": file_output_list,
    "Relevancy Percentage":probability_list
})
print(output_df)
```

---

```
output_df['Sort_Yes'] = (output_df['Relevant?'] == 'Yes').astype(int)
reranked_df = output_df.sort_values(by=['Sort_Yes', 'Relevancy Percentage'
   ], ascending=[False, False])
reranked_df = reranked_df.drop(columns=['Sort_Yes'])
reranked_df = reranked_df.reset_index(drop=True)
reranked_rel_df = reranked_df[reranked_df['Relevant?'] == 'Yes']
reranked_rel_df = reranked_rel_df.reset_index(drop=True)
num_rows = len(reranked_rel_df)
print(f"The number of relevant standards returned for your project is {
   num_rows}")
```

---

```
excel_file_path = "returnedstandardsgptreranked.xlsx"
reranked_rel_df.to_excel(excel_file_path, index=True)
print(f"DataFrame saved to {excel_file_path}")
```

---

```
GENERATION_PROMPT = """You are an Assistant tasked with generating
   detailed product requirements solely based on the content of
   engineering standard documents and a brief description of the product.
   Your task is to filter and adapt those standards that are contextually
   relevant to the product's specific needs and applications. When
   creating these requirements, adhere to the guidelines below:
```

- **Correct Terms Usage**:
  - Use "Shall" for mandatory requirements.
  - Use "Will" for statements of fact or declarations of purpose.
  - Use "Should" for recommendations or goals.

- **Requirements Formulation**:
  - Start each requirement with "The project shall" followed by the specific action or standard to be met, directly relating to the product's context and needs as described.
  - Ensure the use of active voice throughout, clearly stating the required actions.
  - Maintain consistent terminology for the product and its components, as specified in the engineering standard.
  - Requirements must include any specified tolerances and be clear, concise, and devoid of implementation or operational methods.
  - When referencing specific sections of the engineering standard document, include the document filename in the reference (e.g., "as specified in [Document Filename] S5.1").
  - Critically assess the relevance of each section of the engineering standard document to the product description. Only include standards that are contextually relevant to the product's specific needs and applications.

- **Focused on Engineering Standards**:
  - Directly extract and adapt requirements from the engineering standard document, focusing on those aspects that are most relevant to the product's compliance and performance.
  - Emphasize translating the engineering standards into actionable product requirements, rather than deriving them from the product description.

```
- **General Guidelines**:
  - Ensure all requirements are grammatically correct and adhere to the
    project's template and stylistic guidelines.
  - State requirements in a positive manner, avoid "To Be Determined"
    values whenever possible, provide rationale for requirements based on
    the standard document, and logically organize requirements within the
    document.

**Input**:
Product Description: {query}
Engineering Standard Document: {document}
Document Filename: {filename}
**Task**:
Based on the Engineering Standard Document provided and considering the
    Product Description, generate product requirements that are fully
    compliant with the standard. Focus on translating standards into
    specific, actionable requirements for the product that are contextually
     relevant to its specific needs and applications. Ensure these
    requirements are complete and leave no room for ambiguity in compliance
     and implementation. Include the document filename when making section
    references (e.g., "[Document_Filename] S5.1"). Requirements unrelated
    to the product's context may be omitted. YOU SHOULD ONLY RETURN GENERAL
     REQUIREMENTS. NO OTHER DOCUMENT SECTIONS SHOULD BE PRESENT
"""
```

---

```
@retry(wait=wait_random_exponential(min=1, max=40), stop=
    stop_after_attempt(3))
def generate_requirement(query, document,filename):
    response = client.chat.completions.create(
        model=OPENAI_MODEL,
        messages=[{
            "role":
```

```
            "user",
            "content":
            GENERATION_PROMPT.format(query=query, document=document,
    filename=filename)
        }],
        temperature=0,
        max_tokens=1000,
    )

    return response
```

---

```
relevant_retrieved_standards = reranked_rel_df['Filename'].head(num_rows).
    tolist()
print(relevant_retrieved_standards)
relevant_retrieved_standards_content = {}
for filename in relevant_retrieved_standards:
    if filename in files_content_dict:
        relevant_retrieved_standards_content[filename] =
    files_content_dict[filename]
    else:
        print(f"Content for {filename} not found in files_content_dict.")
```

---

```
query = file_content
requirement_output_list = []
standard_source_list = []
output_directory = "SARG Results/"

for filename, content in relevant_retrieved_standards_content.items():
    try:
        base_filename = os.path.splitext(filename)[0]
        output_filename = os.path.join(output_directory, f"{base_filename}
    _SARGv2.txt")
```

```
        output = generate_requirement(query, document=content, filename=
    base_filename)
        requirement_output = output.choices[0].message.content
        requirement_output_list.append(requirement_output)
        standard_source_list.append(filename)

        base_filename = os.path.splitext(filename)[0]
        output_filename = os.path.join(output_directory, f"{base_filename}
    _requirements.txt")

        with open(output_filename, "w") as file:
            file.write(requirement_output)

    except Exception as e:
        print(f"Error processing document {filename}: {e}")
```

---

```
output_df = pd.DataFrame({
    "Requirements": requirement_output_list,
    "Filename": standard_source_list,
})
```

---

```
folder_path = output_directory
output_file_path = "SARG Results/combined_requirements_output.txt"
files = os.listdir(folder_path)

with open(output_file_path, 'w') as outfile:
    for filename in files:
        if filename.endswith('.txt'):
            file_path = os.path.join(folder_path, filename)
            outfile.write(f"Filename: {filename}\n\n")
            with open(file_path, 'r') as infile:
                contents = infile.read()
                outfile.write(contents + "\n\n")
```

```
print(f"All text files have been combined into {output_file_path}")
```

---

```
with open(output_file_path, 'r') as file:
    RequirementFile=file.read()
```

---

```
COMPILATION_PROMPT = """You are an Assistant tasked with compiling and
   reviewing a draft requirements document. Given the engineering
   requirement document and a product description as inputs, you should
   verify and or achieve the following:

- **Verify Correct Terms Use**:
  - The document should use "Shall" for mandatory requirements.
  - The document should use "Will" for statements of fact or declarations
   of purpose.
  - The document should use  "Should" for recommendations or goals.

- **Requirements Formulation**:
  - The document should Start each requirement with "The Project  shall"
   followed by the specific action or standard to be met, as dictated by
   the engineering standard document.
  - The document should use of active voice throughout, clearly stating
   the required actions.
  - The document should maintain consistent terminology for the product
   and its components, as specified in the engineering standard.

- **Eliminate Redundant Requirements**:
  - Ensure that each requirement in your output is unique in detail.


- **General Guidelines**:
  - Logically organize requirements within the document by type of
   requirement and then by component.
```

```
**Input**:
Product Description: {query}
Engineering Requirement Document: {document}
**Task**:

Based on the Engineering Requirement Document provided, generate a set of
   refined product requirements that are fully compliant with all details
   from the Engineering Requirement Document. Ensure that the requirements
    you generate are high level requirements that consisely summarize all
   of the detail in the input engineering requirement document. The
   document should Start each requirement with "The Project  shall"
   followed by the specific action or standard to be met, as dictated by
   the engineering standard document."""
```

---

```
@retry(wait=wait_random_exponential(min=1, max=40), stop=
   stop_after_attempt(3))
def generate_requirement_document(query, document):
    response = client.chat.completions.create(
        model=OPENAI_MODEL,
        messages=[{
            "role":
            "user",
            "content":
            COMPILATION_PROMPT.format(query=query, document=document)
        }],
        temperature=0,
        max_tokens=4096,
    )

    return response
```

---

```
query = file_content
```

```
document = RequirementFile
output = generate_requirement_document(query, document)
requirement_document_output = output.choices[0].message.content
output_filename_document = f"SARG Results/Results_{OPENAI_MODEL}.txt"

with open(output_filename_document, "w") as file:
    file.write(requirement_document_output)
```

---

```
VERIFICATION_PROMPT = """You are an Assistant tasked with compiling and
   reviewing a draft requirements document. Given the engineering
   requirement document and a product description as inputs, you should
   verify and or achieve the following:

- **Ensure Completeness**:
  - Ensure that the Engineering Requirement Document is fully represented
   in the Final Requirement Document
  - The document should use "Will" for statements of fact or declarations
   of purpose.
  - The document should use  "Should" for recommendations or goals.
    - The document should Start each requirement with "The Project  shall"
    followed by the specific action or standard to be met, as dictated by
   the engineering standard document.


- **Eliminate Redundant Requirements**:
  - Ensure that each requirement in your output is unique in detail.


- **General Guidelines**:
  - Logically organize requirements within the document by type of
   requirement and then by component.
```

```
**Input**:
Product Description: {query}
Engineering Requirement Document: {document}
Final Requirement Document: {final_doc}
**Task**:

Based on the Engineering Requirement Document provided, ensure that all
   detail is represented in Final Requirement Document, while eliminating
   redundancy, in your response.The document should Start each requirement
    with "The Project  shall" followed by the specific action or standard
   to be met, as dictated by the engineering standard document.
"""
```

---

```
@retry(wait=wait_random_exponential(min=1, max=40), stop=
   stop_after_attempt(3))
def generate_requirement_document_verified(query, document, final_doc):
    response = client.chat.completions.create(
        model=OPENAI_MODEL,
        messages=[{
            "role":
            "user",
            "content":
            VERIFICATION_PROMPT.format(query=query, document=document,
   final_doc=final_doc)
        }],
        temperature=0,
        max_tokens=4096,
    )

    return response
```

---

```
query = file_content
final_doc = requirement_document_output
```

```
document = RequirementFile
output = generate_requirement_document_verified(query, document, final_doc
    )
verified_requirement_document_output = output.choices[0].message.content
output_filename_document = f"SARG Results/
    final_combined_requirements_output.txt"

with open(output_filename_document, "w") as file:
     file.write(verified_requirement_document_output)
print(verified_requirement_document_output)
```

# Bibliography

(2022). https://openai.com/blog/new-and-improved-embedding-model

Alonzo, R. J. (2010). Chapter 2 - american versus global. In R. J. Alonzo (Ed.), *Electrical codes, standards, recommended practices and regulations* (pp. 55–77). William Andrew Publishing. https://doi.org/https://doi.org/10.1016/B978-0-8155-2045-0.10002-3

ASME. (2010). The history of asme's boiler and pressure vessel code. https://www.asme.org/topics-resources/content/the-history-of-asmes-boiler-and-pressure

ASME. (2024). *ASME Standards Certification FAQ* [Accessed: March 20, 2024]. https://www.asme.org/codes-standards/publications-information/faq

Beitz, W., Pahl, G., & Grote, K. (1996). Engineering design: A systematic approach. *Mrs Bulletin*, *71*.

Beshoy Morkos, P. S., & Summers, J. D. (2012). Predicting requirement change propagation, using higher order design structure matrices: An industry case study. *Journal of Engineering Design*, *23*(12), 905–926. https://doi.org/10.1080/09544828.2012.662273

Bodner, D., & Rouse, W. (2009). Handbook of systems engineering and management. *Wiley. chapter Organizational Simulation*.

Bouzidi, K. R., Fies, B., Faron-Zucker, C., Zarli, A., & Thanh, N. L. (2012). Semantic web approach to ease regulation compliance checking in construction industry. *Future Internet*, *4*(3), 830–851.

Brants, T., & Franz, A. (2006). Web 1t 5-gram. *Philadelphia, PA: LDC Data Consortium*.

Cai, D., Wang, Y., Li, H., Lam, W., & Liu, L. (2021). Neural machine translation with monolingual translation memory. *Proceedings of the 59th Annual Meeting of the Association for Computational*

*Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, 7307–7318.

Chen, C. (2022). *Realization of inter-model connections: Linking requirements and computer-aided design* (Doctoral dissertation). University of Georgia.

Chen, C., Carroll, C., & Morkos, B. (2023). From text to images: Linking system requirements to images using joint embedding. *Proceedings of the Design Society*, *3*, 1985–1994.

Chen, C., & Morkos, B. (2023). Exploring topic modelling for generalising design requirements in complex design. *Journal of Engineering Design*, *34*(11), 922–940.

Chen, C., Mullis, J., & Morkos, B. (2021). A topic modeling approach to study design requirements. *International design engineering technical conferences and computers and information in engineering conference*, *85383*, V03AT03A021.

Chen, C., Wei, S., & Morkos, B. (2023). Bridging the knowledge gap between design requirements and cad-a joint embedding approach. *2023 ASEE Annual Conference & Exposition*.

Clark, K., Luong, M.-T., Le, Q. V., & Manning, C. D. (2020). Electra: Pre-training text encoders as discriminators rather than generators. *arXiv preprint arXiv:2003.10555*.

Coulson, T. (1944). The origin of interchangeable parts. *Journal of the Franklin Institute*, *238*(5), 335–344.

Cronembold, J. R., & Law, K. H. (1988). Automated processing of design standards. *Journal of computing in civil engineering*, *2*(3), 255–273.

De Jong, J. P., & Marsili, O. (2006). The fruit flies of innovations: A taxonomy of innovative small firms. *Research policy*, *35*(2), 213–229.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018a). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018b). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

DIN. (2024). *A Brief Introduction to Standards* [Accessed: March 20, 2024]. https://www.din.de/en/about-standards/a-brief-introduction-to-standards

Dunn, A., Dagdelen, J., Walker, N., Lee, S., Rosen, A. S., Ceder, G., Persson, K., & Jain, A. (2022). Structured information extraction from complex scientific text with fine-tuned large language models. *arXiv preprint arXiv:2212.05238*.

Elbedweihy, K. M., Wrigley, S. N., Clough, P., & Ciravegna, F. (2015). An overview of semantic search evaluation initiatives [Semantic Search]. *Journal of Web Semantics*, *30*, 82–105. https://doi.org/https://doi.org/10.1016/j.websem.2014.10.001

Esteva, A., Kale, A., Paulus, R., Hashimoto, K., Yin, W., Radev, D., & Socher, R. (2021). Covid-19 information retrieval with deep-learning based semantic search, question answering, and abstractive summarization. *NPJ digital medicine*, *4*(1), 68.

Fenves, S. J. (1966). Tabular decision logic for structural design. *Journal of the Structural Division*, *92*(6), 473–490. https://doi.org/10.1061/JSDEAG.0001567

Firesmith, D. (2003). Specifying good requirements. *Journal of Object Technology*, *2*(4), 77–87.

Galley-Taylor, M., Ferguson, A., & Hayward, G. (2011). Role of standards in design.

Greene, R., Sanders, T., Weng, L., & Neelakantan, A. (2022). New and improved embedding model.

Großer, K., Riediger, V., & Jürjens, J. (2022). Requirements document relations: A reuse perspective on traceability through standards. *Software and Systems Modeling*, *21*(6), 1–37.

Group, I. R. W., et al. (2019). Guide for writing requirements. *INCOSE: San Diego, CA, USA*.

Gu, J., Wang, Y., Cho, K., & Li, V. O. (2018). Search engine guided neural machine translation. *Proceedings of the AAAI Conference on Artificial Intelligence*, *32*(1).

Guo, J., Fan, Y., Ai, Q., & Croft, W. B. (2016). A deep relevance matching model for ad-hoc retrieval. *Proceedings of the 25th ACM international on conference on information and knowledge management*, 55–64.

Haque, M. A., & Li, S. (2023). The potential use of chatgpt for debugging and bug fixing. *EAI Endorsed Transactions on AI and Robotics*, *2*. https://doi.org/10.4108/airo.v2i1.3276

Harrs, S. (2006). *Standards directory: An engineering and technology standards digital library and information retrieval system for the walt disney company* (Doctoral dissertation). https://www.

proquest.com/dissertations-theses/standards-directory-engineering-technology/docview/304909870/se-2

Hauksdóttir, D., Ritsing, B., Andersen, J. C., & Mortensen, N. H. (2016). Establishing reusable requirements derived from laws and regulations for medical device development. *2016 IEEE 24th International Requirements Engineering Conference Workshops (REW)*, 220–228. https://doi.org/10.1109/REW.2016.045

Hein, P. H., Kames, E., Chen, C., & Morkos, B. (2021). Employing machine learning techniques to assess requirement change volatility. *Research in engineering design*, *32*, 245–269.

Hein, P. H., Kames, E., Chen, C., & Morkos, B. (2022). Reasoning support for predicting requirement change volatility using complex network metrics. *Journal of Engineering Design*, *33*(11), 811–837.

Hein, P. H., Menon, V., & Morkos, B. (2015). Exploring requirement change propagation through the physical and functional domain. *International design engineering technical conferences and computers and information in engineering conference*, *57052*, V01BT02A051.

Hein, P. H., Voris, N., & Morkos, B. (2018). Predicting requirement change propagation through investigation of physical and functional domains. *Research in Engineering Design*, *29*, 309–328.

Htet Hein, P., Morkos, B., & Sen, C. (2017). Utilizing node interference method and complex network centrality metrics to explore requirement change propagation. *International design engineering technical conferences and computers and information in engineering conference*, *58110*, V001T02A081.

Jha, N., & Mahmoud, A. (2019). Mining non-functional requirements from app store reviews. *Empirical Software Engineering*, *24*, 3659–3695.

Jiao, X., Yin, Y., Shang, L., Jiang, X., Chen, X., Li, L., Wang, F., & Liu, Q. (2019). Tinybert: Distilling bert for natural language understanding. *arXiv preprint arXiv:1909.10351*.

Joshi, M., Chen, D., Liu, Y., Weld, D. S., Zettlemoyer, L., & Levy, O. (2020). Spanbert: Improving pre-training by representing and predicting spans. *Transactions of the association for computational linguistics*, *8*, 64–77.

Jury, B., Lorusso, A., Leinonen, J., Denny, P., & Luxton-Reilly, A. (2024). Evaluating llm-generated worked examples in an introductory programming course.

Kim, Y., Bang, S., Sohn, J., & Kim, H. (2022). Question answering method for infrastructure damage information retrieval from textual data using bidirectional encoder representations from transformers. *Automation in construction*, *134*, 104061.

Kotonya, G., & Sommerville, I. (1998). *Requirements engineering: Processes and techniques*. Wiley Publishing.

Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., & Soricut, R. (2019). Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*.

Li, H., Su, Y., Cai, D., Wang, Y., & Liu, L. (2022). A survey on retrieval-augmented text generation. *arXiv preprint arXiv:2202.01110*.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., & Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Loshchilov, I., & Hutter, F. (2017). Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

Luhn, H. P. (1957). A statistical approach to mechanized encoding and searching of literary information. *IBM Journal of research and development*, *1*(4), 309–317.

Luhn, H. P. (1958). The automatic creation of literature abstracts. *IBM Journal of research and development*, *2*(2), 159–165.

Luttmer, J., Prihodko, V., Ehring, D., & Nagarajah, A. (2023). Requirements extraction from engineering standards–systematic evaluation of extraction techniques. *Procedia CIRP*, *119*, 794–799.

Maron, M., Kuhns, J., & Ray, L. (1959). Probabilistic indexing: A statistical technique for document identification and retrieval. *Thompson Ramo Wooldridge Inc, Los Angeles, California, Data Systems Project Office, Technical Memorandum*, *3*.

Mcclung, B. (2011). Volunteerism for standards development [standards]. *IEEE Industry Applications Magazine*, *17*(2), 75–80.

McLellan, J. M., Morkos, B., Mocko, G. G., & Summers, J. D. (2010). Requirement modeling systems for mechanical design: A systematic method for evaluating requirement management tools and languages. *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, *44113*, 1247–1257.

Mihany, F. A., Moussa, H., Kamel, A., Ezzat, E., & Ilyas, M. (2016). An automated system for measuring similarity between software requirements. *Proceedings of the 2nd Africa and Middle East Conference on software engineering*, 46–51.

Morkos, B., Joshi, S., & Summers, J. D. (2012). Representation: Formal development and computational recognition of localized requirement change types. *International design engineering technical conferences and computers and information in engineering conference*, *45028*, 111–122.

Morkos, B., Joshi, S., & Summers, J. D. (2019). Investigating the impact of requirements elicitation and evolution on course performance in a pre-capstone design course. *Journal of Engineering Design*, *30*(4-5), 155–179.

Morkos, B., Joshi, S., Summers, J. D., & Mocko, G. G. (2010a). Evaluation of requirements and data content within industry in-house developed data management system. *International Design Engineering Technical Conference*.

Morkos, B., Joshi, S., Summers, J. D., & Mocko, G. G. (2010b). Requirements and data content evaluation of industry in-house data management system. *International design engineering technical conferences and computers and information in engineering conference*, *44113*, 493–503.

Morkos, B., Mathieson, J., & Summers, J. D. (2014). Comparative analysis of requirements change prediction models: Manual, linguistic, and neural network. *Research in Engineering Design*, *25*, 139–156.

Morkos, B., & Summers, J. D. (2010). Requirement change propagation prediction approach: Results from an industry case study. *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, *44090*, 111–121.

Morkos, B., & Summers, J. D. (n.d.). Elicitation and development of requirements through integrated methods. *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, *48999*, 1007–1015.

Morkos, B. W. (2012). *Computational representation and reasoning support for requirements change management in complex system design* (Doctoral dissertation). Clemson University.

Mullis, J., Chen, C., Morkos, B., & Ferguson, S. (2023). Deep Neural Networks in Natural Language Processing for Classifying Requirements by Origin and Functionality: An Application of BERT in System Requirements. *Journal of Mechanical Design*, *146*(4), 041401. https://doi.org/10.1115/1.4063764

NASA. (2024). *Appendix C: How to write a good requirement* [Accessed: March 20, 2024]. https://www.nasa.gov/reference/appendix-c-how-to-write-a-good-requirement/

Neelakantan, A., Xu, T., Puri, R., Radford, A., Han, J. M., Tworek, J., Yuan, Q., Tezak, N., Kim, J. W., Hallacy, C., et al. (2022). Text and code embeddings by contrastive pre-training. *arXiv preprint arXiv:2201.10005*.

NISO. (2017). Sts: Standards tag suite.

Pan, S. J., & Yang, Q. (2009). A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, *22*(10), 1345–1359.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Pennington, J., Socher, R., & Manning, C. D. (2014). Glove: Global vectors for word representation. *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 1532–1543.

Porter, M. F. (1980). An algorithm for suffix stripping. *Program*, *14*(3), 130–137.

Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., et al. (2018). Improving language understanding by generative pre-training.

Rathore, D. B. (2023). Future of ai amp; generation alpha: Chatgpt beyond boundaries. *Eduzone: International Peer Reviewed/Refereed Multidisciplinary Journal*, *12*(1), 63–68. https://www.eduzonejournal.com/index.php/eiprmj/article/view/254

Reimers, N., & Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*.

Robertson, S., & Robertson, J. (2012). *Mastering the requirements process: Getting requirements right*. Addison-wesley.

Rouland, Q., Gjorcheski, S., & Jaskolka, J. (2023). Eliciting a security architecture requirements baseline from standards and regulations. *2023 IEEE 31st International Requirements Engineering Conference Workshops (REW)*, 224–229. https://doi.org/10.1109/REW57809.2023.00045

Salton, G., Wong, A., & Yang, C. S. (1975). A vector space model for automatic indexing. *Commun. ACM*, *18*(11), 613–620. https://doi.org/10.1145/361219.361220

Salton, G. (1962). Some experiments in the generation of word and document associations. *Proceedings of the December 4-6, 1962, fall joint computer conference*, 234–250.

Salton, G., & Yang, C.-S. (1973). On the specification of term values in automatic indexing. *Journal of documentation*, *29*(4), 351–372.

Sanderson, M., & Croft, W. B. (2012). The history of information retrieval research. *Proceedings of the IEEE*, *100*(Special Centennial Issue), 1444–1451. https://doi.org/10.1109/JPROC.2012.2189916

Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019). Distilbert, a distilled version of bert: Smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.

Shabi, J., Reich, Y., Robinzon, R., & Mirer, T. (2021). A decision support model to manage overspecification in system development projects. *Journal of Engineering Design*, *32*(7), 323–345.

Shankar, P., Morkos, B., & Summers, J. D. (2010). A hierarchical modeling scheme with non functional requirements. *International design engineering technical conferences and computers and information in engineering conference*, *44113*, 283–295.

Shankar, P., Morkos, B., & Summers, J. D. (2012). Reasons for change propagation: A case study in an automotive oem. *Research in Engineering Design*, *23*, 291–303.

Shankar, P., Morkos, B., Yadav, D., & Summers, J. D. (2020). Towards the formalization of non-functional requirements in conceptual design. *Research in engineering design*, *31*, 449–469.

Shishko, R., & Aster, R. (1995). Nasa systems engineering handbook. *NASA systems engineering handbook/by Robert Shishko; with contributions by Robert Aster...[et al.]; edited by Randy Cassingham.[Washington, DC?]: National Aeronautics and Space Administration,[1995]*, *6105*.

Sparck Jones, K. (1972). A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, *28*(1), 11–21.

Summers, J. D., Joshi, S., & Morkos, B. (2014). Requirements evolution: Relating functional and non-functional requirement change on student project success. *International design engineering technical conferences and computers and information in engineering conference*, *46346*, V003T04A002.

Taube, M., Gull, C. D., & Wachtel, I. S. (1952). Unit terms in coordinate indexing. *American Documentation*, *3*(4), 213–218. https://doi.org/https://doi.org/10.1002/asi.5090030404

Terry Bahill, A., & Henderson, S. J. (2005). Requirements development, verification, and validation exhibited in famous failures. *Systems engineering*, *8*(1), 1–14.

Tian, J., Zhang, L., & Lian, X. (2023). A cross-level requirement trace link update model based on bidirectional encoder representations from transformers. *Mathematics*, *11*(3). https://doi.org/10.3390/math11030623

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, *30*.

Walsh, H. S., & Andrade, S. R. (2022). Semantic search with sentence-bert for design information retrieval. *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, *86212*, V002T02A066.

Wiegers, K. E., & Beatty, J. (2013). *Software requirements*. Pearson Education.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Le Scao, T., Gugger, S., … Rush, A. (2020). Transformers: State-of-the-art natural language processing. In Q. Liu & D. Schlangen (Eds.), *Proceedings of the 2020 conference on empirical methods in natural language processing: System demonstrations* (pp. 38–45). Association for Computational Linguistics. https://doi.org/10.18653/v1/2020.emnlp-demos.6

Xia, X., Bao, L., Lo, D., Xing, Z., Hassan, A. E., & Li, S. (2017). Measuring program comprehension: A large-scale field study with professionals. *IEEE Transactions on Software Engineering*, *44*(10), 951–976.