Journal on
Artificial Intelligence

Tech Science Press

DOI: 10.32604/jai.2024.058649

ARTICLE

Using Artificial Intelligence Techniques in the Requirement Engineering Stage of Traditional SDLC Process

Afam Okonkwo*, Pius Onobhayedo and Charles Igah

Computer and Information Sciences, Pan Atlantic University, Ibeju-Lekki, Lagos, 73688, Nigeria

*Corresponding Author: Afam Okonkwo. Email: afam.okonkwo@pau.edu.ng

Received: 17 September 2024 Accepted: 04 December 2024 Published: 31 December 2024

ABSTRACT

Artificial Intelligence, in general, and particularly Natural language Processing (NLP) has made unprecedented progress recently in many areas of life, automating and enabling a lot of activities such as speech recognition, language translations, search engines, and text-generations, among others. Software engineering and Software Development Life Cycle (SDLC) is also not left out. Indeed, one of the most critical starting points of SDLC is the requirement engineering stage which, traditionally, has been dominated by business analysts. Unfortunately, these analysts have always done the job not just in a monotonous way, but also in an error-prone, tedious, and inefficient manner, thus leading to poorly crafted works with lots of requirement creep and sometimes technical debts. This work, which is the first iteration in a series, looks at how this crucial initial stage could not just be automated but also improved using the latest techniques in Artificial Intelligence and NLP. Using the popular and available PROMISE dataset, the emphasis, for this first part, is on improving requirement engineering, particularly the classification of Functional and Non-functional Requirements. Transformer-powered BERT (Bidirectional Encoder Representations from Transformers) Large Language Model (LLM) was adopted with validation performances of 0.93, 0.88, and 0.88. The experimental results showed that Base-BERT LLM, its distilled counterpart, Distil-BERT, and its domain-specific version, Code-BERT, can be reliable in these tasks. We believe that our findings could encourage the adoption of LLM, such as BERT, in Requirement Engineering (RE)-related tasks like the FR/NFR classification. This kind of insight can help RE researchers as well as industry practitioners in their future work.

KEYWORDS

NLP; artificial intelligence; software; requirement engineering; SDLC; requirement classification; functional and non-functional requirement; large language model (LLM); BERT

1 Introduction

Artificial Intelligence is currently leading many innovations in our world [1] and making lots of inroads into the software development life cycle, the SDLC [2]. For instance, GitHub's Co-pilot [3] and ChatGPT's ability to generate codes is popular today. Agentic AI and AI agents are becoming popular as shown by frameworks such as AutoGen [4] and MetaGPT [5] due to the agents' abilities to assume and be assigned various roles traditionally reserved for humans.

Copyright © 2024 The Authors. Published by Tech Science Press.
This work is licensed under a Creative Commons Attribution 4.0 International License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

In a typical software process, SDLC, Requirement Engineering (RE) is the first stage. As one of the most crucial stages [6–8], it sets the proper background and foundation for all other stages of the SDLC. Consequently, it facilitates the proper understanding of the whole project regarding its scope and core functionality, and the way it is carried out can predict the success of any software project [9]. Indeed, most failures of software projects are traceable to ineffective approaches at this stage [10].

Undoubtedly, human analysts and other stakeholders play a key role in the requirement elicitation process [11] using natural language.

Unfortunately, there are inherent challenges associated with using these complex human languages. These include issues such as ambiguity, incompleteness, and inconsistencies [12] including the inability to identify Functional Requirements (FR) and Non-functional Requirements (NFR) from requirement texts as articulated by these authors [13].

Some of the aforementioned challenges have been addressed using tools and AI techniques such as NLP [14]. However, most of them involve typical rule-based heuristics, lexical [10] and semi-automatic approaches, information extraction, morphological analysis, conceptual model [14], and other related NLP techniques. These adopted approaches have yielded remarkable results and improvements in solving these problems [14]. However, there still exists a gap, as regards using the latest innovations in NLP, such as advanced embedding techniques and using Transformer architecture, and Large Language Models (LLMs), to better tackle these inherent difficulties found in natural language and requirement engineering processes. This gap is backed by research conducted by Hou et al. [15] who discovered that among all the stages of SDLC, RE has recorded only 4.72% adoption of LLM. Compare this to implementation and maintenance phases that have had 58.3% and 24.89%, respectively.

Therefore, this paper specifically and concretely addresses the above gap: Classification of Functional and Non-functional Requirements from requirement texts by fine-tuning Transformer-based Large Language Model (LLM) specifically BERT.

It suffices to state here that due to time and resource constraints, it was not possible to tackle all these problems at once; however, we believe that this first experiment was able to address the last issue. And there is a plan to attempt other inherent requirement problems in future works.

Using the popular PROMISE NFR dataset [16]. Our experiments showed that BERT-Base particularly outperformed comparably, even on little fine-tuning and reduced training times, with those of the authors [17–19]. Incidentally, our findings also confirmed the report, by the developers of Distil-BERT, that it retains about 95% of the capability of the base BERT counterpart that it was distilled from. We also observed that the finding of this work agrees with the findings of researchers such as Alhoshan et al. [17] that generic LLM outperforms domain-specific ones.

Our results encourage the adoption of LLM, especially BERT, in RE-related tasks such as the classification of FR/NFR. This kind of insight can help RE researchers as well as industry practitioners in their future work.

This paper is structured as follows: Section 2 reports the background and review of related literature, Section 3 details the proposed and adopted methodology, Section 4 explains the experimental results and briefly states the validity threats. Finally, Section 5 outlines the conclusion with some limitations as well as recommendations.

2 Background and Literature Review

2.1 Overview

In this section, a background and a review of the related literature will be provided.

2.2 Background and Related Literature

Artificial Intelligence (AI) is currently being seen at the forefront of driving the 5th generation of the Industrial Revolution. As of the time of writing, some top business executives are calling for a temporary halt in the development of AI. In addition, there are debates currently going on regarding many jobs that are in the process of being displaced by it. Jobs such as content writers, editors, teachers, and even programmers are listed as being on the line. While these debates are still ongoing with arguments for and against them, there is no doubt that AI is causing lots of disruptions in many industries. Many people have also pointed to its risks in general and in the military. A case in point, a top AI chief from Google, Geoffrey Hinton, recently resigned, so that he could freely criticize the apparent risks associated with AI. Elon Musk himself, the owner of Twitter (now X) and Tesla, also pledged to build the so-called "Truth GPT" (whatever that means) to compete with both Microsoft-backed OpenAI's GPT and Google's Bard [3], among others. All these serve to buttress further that AI is causing lots of upheaval and automating a lot of domains traditionally done by human beings.

In the realm of software Engineering, AI is also not left out, ranging from Microsoft-owned GitHub Co-pilot [3], to ChatGPT's code generation using Generative Pre-Trained Model (GPT), Some researchers have used TESTPILOT [20] for automatic unit tests generation using LLM, etc. Stack Overflow recently reported experiments currently being carried out in many areas and, also in Google, such as using AI for code reviews and introducing plugins that could lead to the so-called 'self-healing codes'; that is, the ability of codes already committed to the version control system, being automatically corrected and deployed to production by AI after being reviewed once the build process failed in the CI/CD pipeline.

For developers, it is not a secret that software engineering and development is hard and complex [21]. On one hand, partly due to misalignment and the knowledge gap between business analysts or domain experts, who have expertise in their business domain, and software engineers who are masters of technicality, on the other hand. Thus, to simplify things, it has traditionally followed a cycle known as the Software Development Life Cycle (SDLC). In this cycle, requirement engineering is obviously at the forefront [12]. Unfortunately, it has often been done tediously resulting in inherent issues such as vagueness, and incompleteness [12], among others already listed, these call for an innovative approach. With this background, the next section gives a review of related literature and previous works.

These reviews would be divided into these categories: Linguistic/Grammar Rules, GUI/Diagrammatic Aided Builder Tools, Rule-Based/Parsing/Pattern Matching Heuristic Techniques, and ML Models techniques.

2.2.1 Linguistic and Grammar Rules

These are based on pure analysis and parsing of grammar and linguistic rules such as subject, verb, and objects while leveraging on their relationships. Kuchta et al. [22] proposed a methodology, different from domain ontology and traditional statistical approaches, due to their inherent weaknesses. Instead, they proposed a modern grammatical analysis tool powered by the OpenNLP framework and WordNet lexical database. The procedures involved splitting, with the help of OPENLP, the texts into sentences, followed by tokenization, part of speech (POS) tagging, disambiguation of POS, phrase detection, and finally noun phrases. The above steps are combined into an NLP pipeline.

Then concepts are detected with the help of WordNet. Finally, hypernyms, synsets, and hyponyms are used to detect relationships between words. Though their contribution helped to solve the problem of ambiguous concepts, it was not exhaustive since their next action plan, as of the time they drafted the report, was to generate class diagrams from the disambiguated concepts.

2.2.2 GUI-Diagrammatic Aided Tools

In this category, there is usually a graphical user interface (GUI) that enables or aids the analysts in requirements elicitation and management. According to Ibrahim et al. [23], a web-enabled platform called Circle was developed by authors Ambriola and Gervasi for the selection, elicitation, and validation of software requirements. It was both capable of information extraction and measurement of consistency of extracted information. The above authors Ibrahim et al. [23] also built a desktop and GUI-based RACE system, which was able to assist software developers and requirement analysts in generating UML class diagrams; it, however, had the limitation and could not identify one-to-one, one-to-many and many-to-many relationships.

2.2.3 Rule-Based-Parsing and Pattern-Matching Heuristic Technique

For this approach, certain patterns or keywords, entities, relationships, and co-references are extracted from the requirement texts; these, in addition, include the use of heuristic rules. According to Gamage [24], recent studies in these areas use part-of-speech (POS) tagging, domain-centered databases, domain ontology, and entities. According to Ibrahim et al. [23], there was a proposed technique that uses domain ontology and NLP; since the main classes are usually interrelated via distinct types of relationships such as one-to-one, many-to-one, etc. The Object Oriented(OO) classes are discovered via part of speech tagging, grammar parsing, and linguistic patterns, finally, refinement is done using domain ontology. These authors [23] incorporated another heuristic-based approach that they called the Taxonomic Class Model (TCM), and it involves several modeling rules including analysis of nouns, using structural rules of English sentences, and class categories among other heuristic rules. Though this rule-based approach generally worked, however, as is the case with heuristics rules, they didn't cover extensive cases as shown by the CM-Builder system of Herchi et al. [25], this is unlike LLMs which are generally pre-trained with extensive datasets.

2.2.4 Machine Learning and Modern Techniques

ML models are typically trained to extract relationships and entities from NL texts through algorithmic training; this helps to recognize patterns in the context of the texts. According to Gamage [24], these models are good at identifying links and relationships that exist among components. Such popular ML algorithms, used, include Support Vector Machine (SVM), Tree-based algorithms such as the Decision Tree. They made use of initially defined keywords stored in the database, pertaining to a particular domain, with the aid of domain ontology. Also, the same writer Gamage [24] reported that Abdelnabi et al. [12] made use of NLP in combination with heuristic rules to extract UML artifacts, with the aid of the Stanford CoreNLP library. This framework aided in achieving dependency parsing, extraction of information, and tokenization. Logistics and Perceptron classifiers were, furthermore, used. In other works, such as Arachchi [26], the Naive Bayes classifier was used for the attributes and key terms needed to generate Unified Modeling Language (UML) components. In other areas, advanced algorithms such as Convolutional Neural Network (CNN) and Generative Adversarial Network (GAN) were used in the generation of architectural diagrams. ML technique is certainly a modern approach to solving some highly structured data problems; however, it is not as efficient as a deeply contextualized transformer approach such as BERT which uses bi-directional LSTM and can

consider left-to-right contexts. Moreover, its supervised method requires extensive data as shown in the works of Alhosan et al. [17].

2.3 Detailed Literature Review

In the works of Deeptimahanti et al. [27], adopting an NLP, Rule-Based, mostly Model Driven Engineering (MDE) approach, the researchers proposed a domain-independent system called UMGAR, aimed at assisting developers in generating UML analysis, collaboration, and design class models from natural language based-requirement texts. The architecture is broadly divided into two components: the NLP Tool Layer and Model Generator. Some of the key attributes of UMGAR include First, the system used NLP technologies such as Java-RAP, WordNet, Stanford Parser, and XMI import facility for visualization of the generated UML artifacts. Furthermore, it uses a glossary, and eight syntactic reconstruction rules to solve the problem of incompleteness, ambiguities, and other related communication gaps. Also, this technique used some features to identify the context of a particular OO element. Lastly, UMGAR can generate Java code for the corresponding design class model that was generated, in addition to a feature that allows traceability between the requirement texts and generated code.

Ammar et al. [13] surveyed the applications of AI in the software development process, especially SDLC. They presented the available trending tools, for software engineers, industry practitioners, and SDLC process, to focus on instead of the prevailing academic tools of that time. At the same time, they highlighted areas of research. They articulated problems arising from the requirement engineering phase of SDLC such as ambiguity of requirements, incomplete, imprecise, vague, conflicting, and volatile requirements. Finally, they highlighted others including communication problems among stakeholders and difficulty in requirements management.

For Herchi et al. [25], adopting NLP, Domain Ontology, and Heuristics, the scientists implemented a similar approach as the afore-mentioned RACE system [23]. In addition, they introduced XML and used a text-processing java-based open-source tool, *GATE* framework, built by the University of Sheffield. They, in addition, used linguistic, heuristic rules, and just as the previous authors they used domain ontology for refining the identification of concepts. Their solution was, thus, able to identify relationships among OOP objects.

Kuchta et al. [22] proposed a methodology different from domain ontology and traditional statistical approaches due to their inherent weaknesses. They proposed a grammatical analytic tool powered by OpenNLP and WordNet. The procedures involved, using OpenNLP, splitting the texts into sentences, followed by tokenization, part of speech (POS) tagging, disambiguation of POS, phrase detection and finally noun phrases analysis. The above steps are combined into an NLP pipeline. Then, concepts are detected with the help of WordNet. Finally, hypernyms, synsets, and hyponyms are used to detect relationships between words.

Stol et al. [28] tried to solve the problem of inconsistency of research methods and terminologies used in Software Engineering (SE) among researchers, the authors proposed the ABC framework which describes generalizability over Actors(A), an exact measurement of their Behaviors (B) in a realistic Context (C). According to them, this offers a holistic understanding of eight archetypal research strategies. The framework uses two important aspects in research design: *unobtrusiveness/intrusiveness* of the research and *generalizability* of research discoveries. Finally, the newly proposed research strategies were demonstrated in two popular SE domains: Requirement Engineering and Global Software Engineering.

For their part, Wagner et al. [6] observed that there was little or no sound theory guiding SE and RE, the researchers designed a survey instrument that was initially tested in Germany but was later validated and carried out in ten countries, with respondents coming from 228 organizations. Their aim was to establish an empirically based descriptive and explanatory theory for Requirement Engineering (RE). One of their key findings is that interviews, prototyping, and meetings are the most frequently used requirement elicitation processes.

Elallaoui et al. [29] adopted agile, scrum, methodology, and NLP to automatically transform user stories into UML use-case diagrams.

Karunagaran [9] proposed AI and NLP techniques that can be helpful in minimizing human involvement in the requirement stage of SDLC and improving this phase before moving to the design stage. His suggested solutions include, apart from the traditional sentence tokenization approach, modern techniques such as lemmatization, text pre-processing, feature extraction, regex, and classification using ML algorithms (such as Logistic Regressions, Random Forest, Decision Trees, etc.). He, likewise, described how functional and non-functional requirements can be extracted from requirement texts with higher accuracy.

Regarding Abdelnabi et al. [12], the authors proposed a way to generate UML class models from NL texts using NLP and a handful of heuristic rules.

Ameller et al. [30], in their surveys, studied the degree of adoption of Non-Functional Requirements in the context of Model Driven Development (MDD). They interviewed practitioners from 18 companies in 6 European countries. Their findings showed that there is little support for NFRs (Non-Functional Requirements) by practitioners of MDD. Productivity, reusability, and maintainability with expectations met for productivity and maintainability when MDD is adopted.

As regards Lano et al. [31], having discovered an innovative technique to combine metamodel matching with automated model transformations (MT) requirements analysis, these authors worked on improving MT using techniques of Model Transformations by Example (MBTE) and automated Requirement Analysis with the aid of NLP and Machine Learning (ML), The authors used the MBTE to address the first two sets of limitations of meta-model matching. While NLP and ML were used to address the third and fourth sets of identified limitations of the same meta-model matching.

Budake et al. [19] did a literature review and proposed a theoretical approach to discover and classify functional and non-functional requirements of SRS. Their aim was to help software developers and testers to achieve this and improve the creativity of their craft. They described an approach by a researcher that classified requirements text into functional and non-functional requirements using a supervised ML binary classifier algorithm. The experiment produced a recall of 92% using *quite modern* NLP data pre-processing features such as *n-grams* and *bags of words*. However, according to the experiment, higher recall and lower precision were obtained on automatic feature selection. They listed issues inherent in natural language requirements as: Ambiguity, issue of vagueness, incompleteness, subjectivism, and missing elements just as noted by previous authors Ammar et al. [13]. Lastly, following their discussions, the authors recommended the following should be done, by anyone interested in this area, before using NLP in the requirement-gathering stage of SDLC: Domain knowledge and modeling, appreciation of common vocabulary, and clear internalization of a domain, identification of objects, conditions, events, response, state and relations of the system and ordinarily the ML model should receive historical data.

With respect to Franch et al. [32], with the aim of investigating any alignment between RE research and industrial practice, the paper carried out an empirical study via a questionnaire-based approach

to find out the relevance of RE research for practitioners in the industry. It surveyed 435 RE research papers found in major conferences. The study participants were 153, providing 2164 ratings.

Hossain et al. [14], in this work, the researchers extensively surveyed the different traditional and popular NLP-powered conceptual modeling frameworks. Specifically, they studied how each system was constructed, the architecture, motivations, and verification capability, among others.

Hidellaarachchi et al. [33] studied the influence of human aspects such as motivation and personality that impact practitioners' work regarding RE-related activities. The study was in the form of a survey that was carried out on 111 software practitioners of RE-related activities. Their findings suggest that *human aspects* such as motivation, communication, domain knowledge, personality, and attitude are incredibly important in RE. Emotions, cultural diversity, and geographic distributions were found to be moderately important.

Alhoshan et al. [17], using the zero-shot Learning method as well as Transformer-based LLM, the researchers observed that the scientific community had been focusing on using supervised learning techniques, which has the inherent problems of relying exclusively on labeled data, as regards solving Requirement Engineering (RE) challenges. Unfortunately, these annotated data are often lacking or grossly inadequate. Hence, these scientists proposed another approach for RE classifications based on zero-shot learning that does not need labeled data.

Subahi [7], with a noble intention of contributing to a green and sustainable RE and thus greener software, presented a proof-of-concept to having a greener RE. The study also expanded the popular PROMISED dataset to include sustainable elements in the NFR. Finally, BERT LLM was used in this research.

Aranda et al. [11] wanted to confirm the popular assumption that experience improves the elicitation effectiveness of requirement analysts. However, their quasi-experiment proved that the impact of experience on a requirement analyst's effectiveness varies depending on the problem domain. Positive effects were more noticed as regards *interviews and requirement* experiences in familiar domains. They concluded that *experience* has a bearing on analyst effectiveness depending on whether the problem domain is familiar or not.

Hou et al. [15] in their work provided a systematic review of Large Language Models (LLMs) in software engineering, categorizing LLMs used in SE tasks and data collection methods commonly employed. Importantly, the paper examined specific SE tasks where LLMs have been employed most. They discovered that among all the stages of SDLC, RE has recorded 4.72% adoption of LLM. Compare this to the Implementation and Maintenance phases which had 58.3% and 24.89%, respectively.

Lastly, Shreta and Santo [18] explored how AI techniques can be used to improve the design phase of the SDLC.

2.4 Observations from Literature Review

Having reviewed existing literature, a gap is currently missing: Using advanced embedding techniques to improve the requirement process of SDLC. Little or very few researchers seem to have used advanced and deeply contextualized Large Language Models (LLMs) to carry out tasks such as *Requirement Classifications* [15]. This means this novel technique is yet to be fully and sufficiently explored. Hence, this work will explore this approach/methodology to contribute to and advance this field.

3 Methodology

For this chapter, the sections are divided into a brief Statement of the proposed Methodology and Technical Approach, Proposed Architecture, Reason for the Proposed Architecture/Technical Approach, Brief Description of Architectural Components, Proposed Technologies, Frameworks/Libraries, and Configuration of Training Environment, This is followed by Description of Datasets and its Source, Description of Algorithmic Steps, Evaluation Metrics/Hyper Parameter Tuning and lastly Ethical Considerations.

3.1 Research Design and Statement of Methodology and Proposed Technical Approach

Adopted Approach: Fine-Tuning of Bidirectional Encoder Representations from Transformers (BERT) LLM and Attachment of Classification Head to the Transformer-Based Model

Classification of Requirements Texts into Functional (FR) and Functional (NFR) requirements is a popular aspect of Requirement Engineering. This helps both the system analysts, developers, and other stakeholders to have a clearer picture of the new system, and also to know areas to focus initial attention. So, this project intends to fine-tune a pre-trained LLM Model for Functional and Non-Functional Requirements Classifications. The LLM that it will be using is transformer-based BERT LLM, introduced by Google in a paper by Devlin et al. [34]. It will also make use of the following technologies, along with the LLM: Google TensorFlow and its Keras API, Hugging Face Transformer API, and finally both TensorFlow and Hugging Face Hubs to download the pre-trained models. The pre-trained BERT Models that will be used are:

1. *Bert-Base*: This is one of the original versions of BERT trained by Google with 110 million parameters, as reported in detail by Devin et al. [34] and summarized by Okonkwo et al. [35].
2. *Code-BERT*: A version of BERT trained and tailored for both programming texts and language texts, since we are trying to solve software development RE problems, this explains why we adopted it.
3. *Distil-BERT*: Another version of BERT that is lesser in size than the original BERT base model, but nevertheless, as confirmed by Sahn et al. [8], retains about 95%–97% of the original base counterpart's capabilities while being faster by 60%. It is also reported by the same authors to have reduced the total parameters of the BERT base by 40%. Moreover, according to Alhosan et al. [17], Distil-BERT gave the best result in security requirement classification, these key features of the Distil-BERT made us adopt it.

3.1.1 Proposed System Architecture

The proposed overall system architecture is shown in Fig. 1

Note: Fig. 1 summarizes the overall system: typically, the PROMISED NFR datasets are inputted into BERT LLM, and the output is a BERT NFR/FR Classifier for Requirement Gathering.

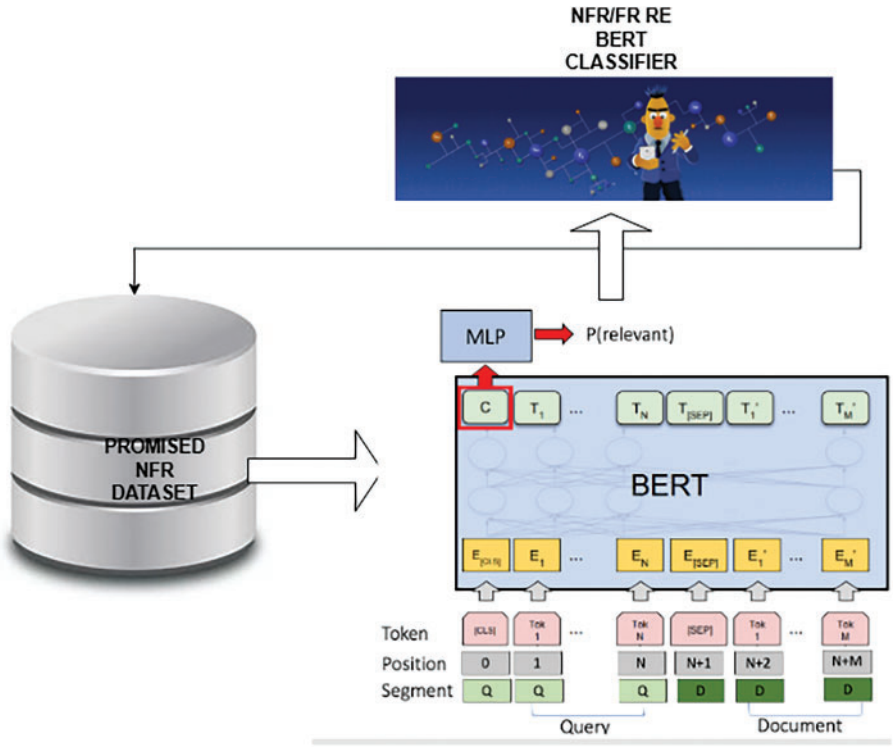


Figure 1: The complete system architecture

Going forward the overall architectural components will be broken down, starting with LLM, followed by justifications for using BERT, TensorFlow, and others, and we will give a brief justification for using these technical components of the architecture:

3.1.2 Justifications for the Proposed Architectural Components and Technical Approach

In the first place, LLM was used for the following reasons:

1. LLM is a pre-trained model, meaning lots of datasets were used during the pre-training. One just needs to fine-tune the weights with little datasets. Since there are insufficient datasets in the software domain, and in particular requirement engineering, this thus makes LLM a perfect candidate.
2. Pre-trained LLM is efficient in terms of energy, reduces computing costs, and carbon footprints, and thus leads to a sustainable and safer climate, apart from efficiency of time and other resources during training.

Secondly, BERT was adopted for the following reasons:

I) BERT is reported to be particularly good in software domains [17], this includes training a classifier in SRS documents [7].

II) It has lots of pre-trained extensions capable of performing a wide array of tasks. Currently, it has at least 3 LLMs available in Hugging Face's NLP hub, capable of performing software-related tasks.

III) BERT is also a deeply contextualized LLM since it is built on top of already existing advanced embedding techniques such as:

1. Transformer: This provides it with the necessary encoder and decoder stacks apart from multi-head attention. This was dealt with in detail in the following sessions.
The reader is advised to read the second part of this work [35] or better still read *Attention is all you need* by Vaswani et al. [36].
2. ULM-FiT: Universal Language Model Fine-Tuning (ULM-FiT) is both an architecture and efficient transfer learning method that can be applied to NLP tasks. It has a method that makes LLMs fine-tunable for different future downstream tasks. It was introduced by Howard and Ruder in a 2018 paper titled *Universal Language Model Fine-tuning for Text Classification*.
3. ELMO: This in turn uses bidirectional LSTM (Long Short-Term Memory) to understand the context of each word.

It makes use of deep contextualized representations of each word based on the other words in the sentence using a type of neural network called bi-directional long short-term memory (LSTM). Unlike BERT, however, ELMO considers the left-to-right and right-to-left paths independently rather than a single unified view of the whole context.

Since context, precision, and ambiguity detection are particularly important in software requirement analysis, the above constituent features make BERT a desirable choice.

Finally, Google TensorFlow and Hugging Face were chosen for the following reasons:

1. Google, apart from being the original developer of BERT, has a sizable number of pre-trained BERT models on its TensorFlow hub.
2. Hugging Face has an even more extensive and larger collection of different ML models, including BERT pre-trained LLMs. The platform, especially, provides lots of software-specific LLMs, some of which are already listed above.

Configuration of Training Environment:

The training environment is made up of the following:

1) Google Colab-It is a popular environment for training ML models and carrying out other data science workflows. The availability of a Graphical Processing Unit (GPU) and Tensor Processing Unit (TPU) makes this environment an attractive option for resource-intensive training.
2) Installing and importing TensorFlow ML Libraries on Colab notebook.
3) Installing and importing, on Google Colab notebook, the afore-mentioned Hugging Face Transformer Framework, and other related libraries.

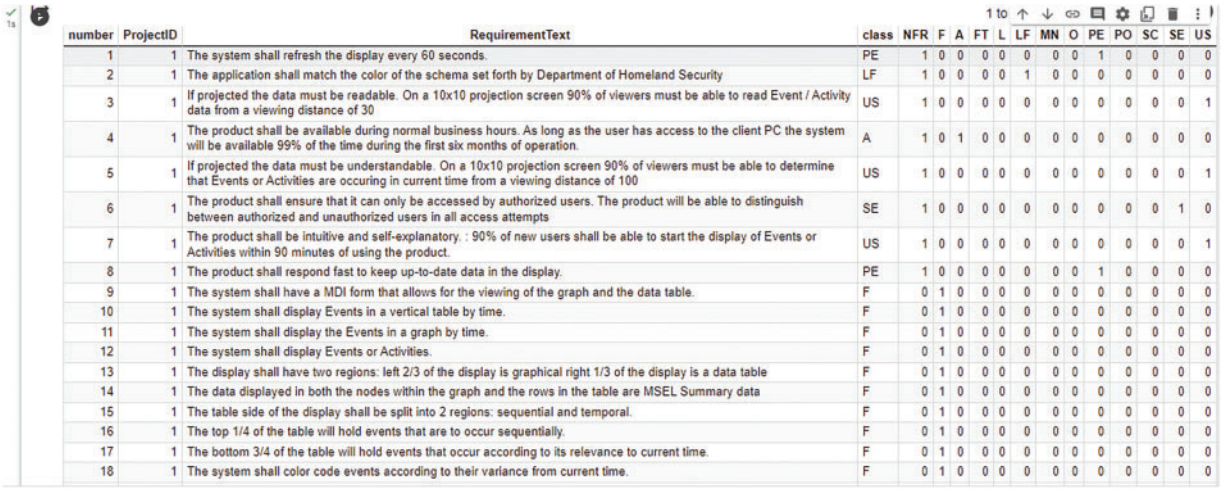
3.1.3 Description of Dataset and Its Source

Since this is a software requirement task, the dataset that will be used is the PROMISE NFR dataset, made available here [16], a popular software requirement dataset widely used in Requirement Engineering by researchers. This dataset [16] is quite small containing only 625 requirement entries. In addition, it is imbalanced since the Non-Functional Requirements (NFRs) class is quite dominant, 59% or 370 while Functional Requirement texts contain 41%, with 255 entries to be precise.

Furthermore, NFRs are divided into 11 classes: Availability, denoted by A, and made up of 21 entries. Legal denoted by L consists of 13 entries, Look and Feel denoted as LF is 38, Maintainability denoted as MN is 17, Operational denoted as O is 62, PE represents Performance and this contains 54 entries, Scalability (SC) is 21, SE represents Security consisting of 66 requirements, US represents

Usability and is totaled 67, Fault Tolerance (FT) has just 10, and Portability denoted as PO is the least with just only 1 requirement belonging to its class.

Fig. 2 shows the first 10 rows of the dataset as displayed by the Data table of Google Colab. The Excel screenshot is also shown.



number	ProjectID	RequirementText	class	NFR	F	A	FT	L	LF	MN	O	PE	PO	SC	SE	US
1	1	The system shall refresh the display every 60 seconds.	PE	1	0	0	0	0	0	0	0	1	0	0	0	0
2	1	The application shall match the color of the schema set forth by Department of Homeland Security	LF	1	0	0	0	0	1	0	0	0	0	0	0	0
3	1	If projected the data must be readable. On a 10x10 projection screen 90% of viewers must be able to read Event / Activity data from a viewing distance of 30	US	1	0	0	0	0	0	0	0	0	0	0	0	1
4	1	The product shall be available during normal business hours. As long as the user has access to the client PC the system will be available 99% of the time during the first six months of operation.	A	1	0	1	0	0	0	0	0	0	0	0	0	0
5	1	If projected the data must be understandable. On a 10x10 projection screen 90% of viewers must be able to determine that Events or Activities are occuring in current time from a viewing distance of 100	US	1	0	0	0	0	0	0	0	0	0	0	0	1
6	1	The product shall ensure that it can only be accessed by authorized users. The product will be able to distinguish between authorized and unauthorized users in all access attempts	SE	1	0	0	0	0	0	0	0	0	0	0	1	0
7	1	The product shall be intuitive and self-explanatory. : 90% of new users shall be able to start the display of Events or Activities within 90 minutes of using the product.	US	1	0	0	0	0	0	0	0	0	0	0	0	1
8	1	The product shall respond fast to keep up-to-date data in the display.	PE	1	0	0	0	0	0	0	0	1	0	0	0	0
9	1	The system shall have a MDI form that allows for the viewing of the graph and the data table.	F	0	1	0	0	0	0	0	0	0	0	0	0	0
10	1	The system shall display Events in a vertical table by time.	F	0	1	0	0	0	0	0	0	0	0	0	0	0
11	1	The system shall display the Events in a graph by time.	F	0	1	0	0	0	0	0	0	0	0	0	0	0
12	1	The system shall display Events or Activities.	F	0	1	0	0	0	0	0	0	0	0	0	0	0
13	1	The display shall have two regions: left 2/3 of the display is graphical right 1/3 of the display is a data table	F	0	1	0	0	0	0	0	0	0	0	0	0	0
14	1	The data displayed in both the nodes within the graph and the rows in the table are MSEL Summary data	F	0	1	0	0	0	0	0	0	0	0	0	0	0
15	1	The table side of the display shall be split into 2 regions: sequential and temporal.	F	0	1	0	0	0	0	0	0	0	0	0	0	0
16	1	The top 1/4 of the table will hold events that are to occur sequentially.	F	0	1	0	0	0	0	0	0	0	0	0	0	0
17	1	The bottom 3/4 of the table will hold events that occur according to its relevance to current time.	F	0	1	0	0	0	0	0	0	0	0	0	0	0
18	1	The system shall color code events according to their variance from current time.	F	0	1	0	0	0	0	0	0	0	0	0	0	0

Figure 2: A view of promise datasets from Google Colab's data table API

As can be seen in Fig. 2, the class label represents different classes of the NFR already listed above. Our target column, the NFR column, is a binary column we want to predict, where a one (1) represents that the requirement text is Non-Functional (NFR), but a zero (0) denotes functional requirement.

3.1.4 Brief Description of Architectural Components, Frameworks and Libraries

The reader is encouraged to read the second part of this work Okonkwo et al. [35] for more details on this section.

3.1.5 Description and Outlining of Algorithmic Steps

In this section, there will be a brief description and outlining of the algorithmic steps followed in the Fine-Tuning of BERT for Requirement Classification.

First, the solution involves two approaches:

1. Using TensorFlow/Keras, via its Data API, Pipeline, and other related models and frameworks: This approach is longer and more complex.
2. Using Hugging Face's Transformer as well as TensorFlow/Keras API: This approach is shorter and less complex.

The first approach involves carrying out the following Tasks:

1. Task 1: *Setting up TensorFlow and Colab Runtime*

2. Task 2: *Load the PROMISE NFR Dataset and carry out some Exploratory Data Analysis (EDA)*. Split the dataset into 80% training and 20% test sets. Furthermore, split the dataset into 75% training sets and 25% validation sets, applying a stratified random sampling technique as an argument in the *train_split* function of the *sklearn* library

3. **Task 3:** *Create tf. data.Datasets for Training and Evaluation*
4. **Task 4:** *Download a Pre-trained BERT Model from TensorFlow Hub*
5. **Task 5:** *Tokenize and Pre-process Text for BERT*
6. **Task 6:** *Wrap a Python Function into a TensorFlow op for Eager Execution*
7. **Task 7:** *Create a TensorFlow Input Pipeline with tf.data*
8. **Task 8:** *Add/Attach Classification Head to BERT*
9. **Task 9:** *Fine-Tune BERT for NFR Classification*
10. **Task 10:** *Evaluate the BERT NFR Classification Model*

The second algorithmic approach, as illustrated in Fig. 3, consists of smaller set of tasks and involves using Hugging Face Transformer and related libraries.

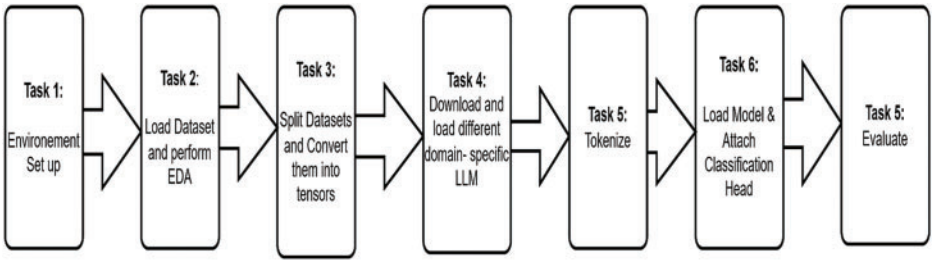


Figure 3: An illustration of the second algorithmic approach

Fig. 3 shows a process diagram illustration of the second algorithmic step.

They are explained in detail as follows:

1. **Task 1:** *Setting up TensorFlow, Google Colab Runtime and Hugging Face Libraries*
2. **Task 2:** *Load the PROMISE NFR dataset and carry out some Exploratory Data Analysis (EDA). Split* the dataset into 80% training and 20% test sets. Furthermore, split the dataset into 75% training sets and 25% validation sets applying a stratified random sampling technique as an argument in the *train_split* function of the *sklearn* library
3. **Task 3:** *Create tf.data.Datasets for Training and Evaluation*
4. **Task 4:** *Download and Load Different Versions of Pre-trained BERT Models from Hugging Face Repo*: In this step, we shall download pre-trained and afore-listed LLMs such as: Sentence BERT, BERT All Mini, Distil-BERT, Hugging Face's Code-BERT, and Stack Overflow's BERT Overflow
5. **Task 5:** *Tokenize Each Domain Specific Model*
6. **Task 6:** *Load Model and Attach Classification Head*
7. **Task 7:** *Evaluate the BERT NFR Classification Model*

3.1.6 Brief Description of Architectural Components, Frameworks and Libraries

For the two approaches, we shall be using binary accuracy since this is the case of binary classification.

3.1.7 Evaluation Metrics and Training Parameters

For the two approaches, we shall be using binary accuracy since this is the case of binary classification. In addition, we shall compare both the validation and the training accuracy metrics. This method helps in finding out if there is a presence of overfitting, known as high variance, or underfitting also called high bias. Other optimization techniques that we plan to use include:

1. Lower epoch values, say 12, since BERT is large with 109 million+ parameters. Reducing the epoch will reduce computing time, though it might affect performance. So, there is a need for a certain level of trade-off.
2. My loss function is BinaryCrossEntropy for the simple reason that this is a fine-tuned BERT model that does binary classification.
3. We shall make use of Adam as an Optimizer for its proven efficiency and popularity. Moreover, it is a viable alternative to stochastic gradient descent since it can adapt the learning rate based on the historical gradient descent.
4. We will, also, make use of other call-back functions from Keras such as ***ReduceLROnPlateau***. This adjusts and reduces the learning rate to 0.0001, for instance, based on the current performance during the training process, ***EarlyStopping*****,** stops the training process when the model performance is no longer improving. This prevents waste of resources and computer time.

3.2 Ethical Considerations

For the PROMISE dataset used in this fine-tuning work, there is no ethical issue, either as regards the acquisition, the processing of the dataset, or the subsequent experiment. Also, the dataset is freely available and has been used extensively by the research community [17]. So, there is nothing affecting ethical concerns such as user privacy or other questions posed by moral philosophers, especially *concern for others, fairness* among other fundamental ethical principles and virtues. However, we have read reports accusing BERT of bias though this is yet to be seen in existing literature. Nevertheless, the prospective user is warned to be careful when using BERT LLM, proposed as the default LLM in this work, as such discrimination could be engraved unconsciously in the massive dataset it was trained on. Also, LLMs are known to perpetuate and amplify biases prevalent in their training data. A case in point: Some researchers [37] carried out a study that revealed gender bias in LLM-generated reference letters. There are also popular cases of hallucination bias.

4 Results Discussion

4.1 Observations from the First Approach: BERT-Base with Google Hub

First, we would like to present, pictorially, outputs as well as Keras plots for this approach. Figs. 4 and 5 both show experimental outputs.

Fig. 4 shows a tabular ***summary*** of input at different layers: BERT Input Layer, Keras Layer, and Output Layers. More importantly, it shows the total number of trainable parameters which is 768 in this case.

Fig. 5 shows a ***visual plot*** of input at different layers: BERT Input Layer, Keras Layer, and Output Layers. More importantly, it shows the total number of trainable parameters which is 768 in this case.

With an epoch value in the range of 4–5, the experimental result is as follows:

1. **Training Accuracy**: 99%
2. **Validation Accuracy**: 93%

```
model_base = create_model();
model_base.compile(optimizer=tf.keras.optimizers.Adam(learning_rate =2e-5),
              loss=tf.keras.losses.BinaryCrossentropy(),
              metrics = tf.metrics.BinaryAccuracy());
model_base.summary();
```

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
==================================================================================================
 input_word_ids (InputLayer  [(None, 150)]             0         []
 )

 input_mask (InputLayer)     [(None, 150)]             0         []

 input_type_ids (InputLayer  [(None, 150)]             0         []
 )

 keras_layer (KerasLayer)    [(None, 768),             1094822   ['input_word_ids[0][0]',
                              (None, 150, 768)]        41         'input_mask[0][0]',
                                                                  'input_type_ids[0][0]']

 dropout (Dropout)           (None, 768)               0         ['keras_layer[0][0]']

 output (Dense)              (None, 1)                 769       ['dropout[0][0]']

==================================================================================================
Total params: 109483010 (417.64 MB)
Trainable params: 109483009 (417.64 MB)
```

Figure 4: TensorFlow-keras model summary of trainable parameters and model size

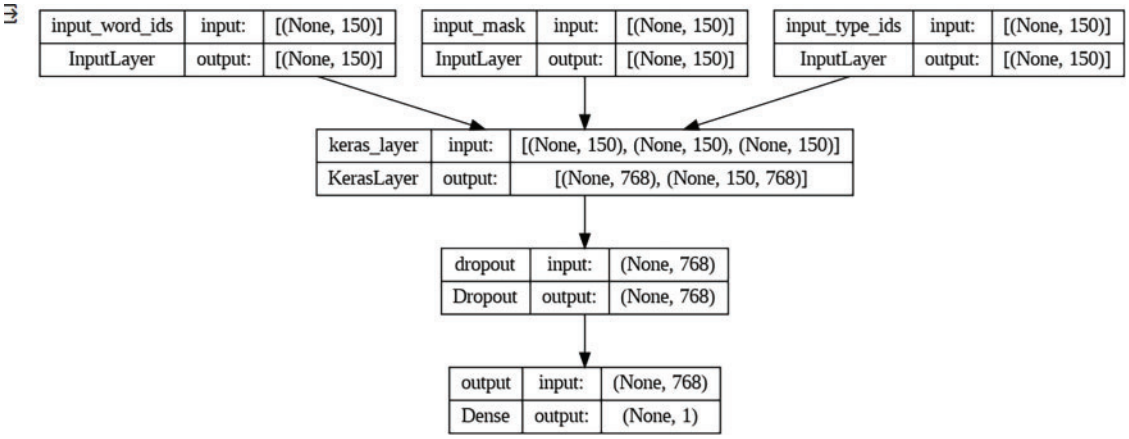


Figure 5: Keras visual plot of trainable parameters

First, in comparison with performances of classical ML algorithms, such as SVM, Naïve Bayes, etc., reported by Alhosan et al. [17], this result demonstrates the potential of the transformer-based model in Requirement Engineering tasks.

However, there is obviously the case of overfitting, even if there is no underfitting. BERT, and in general many large deep learning models, are prone to overfitting, this is because they are usually complex have large trainable parameters and in our case, our PROMISE dataset is quite small (barely 600 data points), this is often one the reasons behind overfitting. Nevertheless, we implemented other strategies for reducing overfitting such as adopting an early stopping technique, and tuning the dropout layer. Lastly, we believe that further tuning or hyperparameter tuning can reduce the overfitting especially if we increase the epoch value which is between 4–5 in this case. Other techniques such as conventional data augmentation, regularization, and training only a few layers of BERT while freezing others could help reduce the overfitting.

4.2 Observations from the Second Algorithmic Approach: Domain-Specific LLM

Please find Figs. 6 and 7 plots of respective training-validation loss and training-validation accuracy for the Code-BERT model. It is plotted against the epoch.

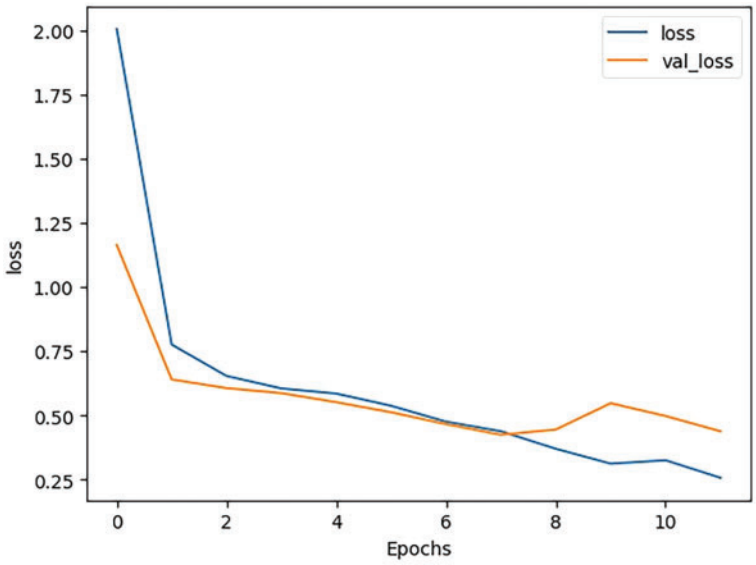


Figure 6: Training-validation loss plotted against epoch

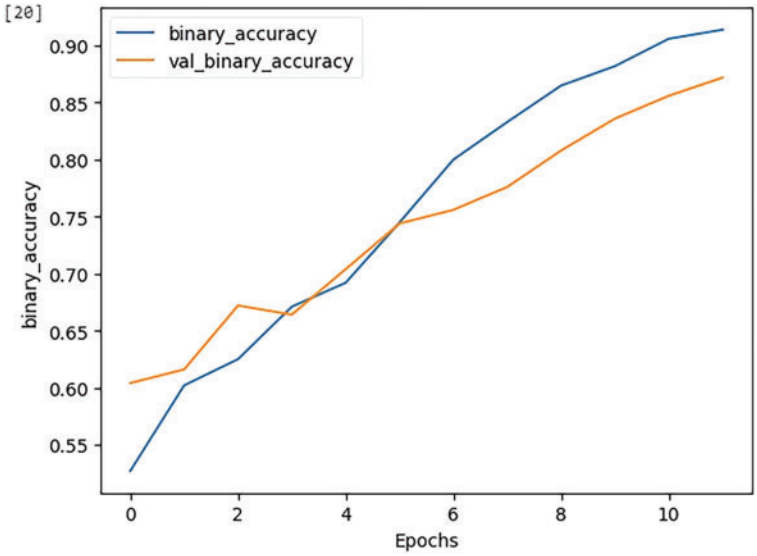


Figure 7: Training-validation accuracy plotted against epoch

Fig. 6 shows that as the training time increases, the calculated loss function result reduces, which means that the model is performing better. At epoch 2.5 and 6.7, the results for both validation and training seem to converge.

Mathematically, we know that for a given input x, and a labeled true output y, the model is expected to predict an output $\hat{y}$.

The loss is supposed to show a discrepancy between y and ŷ.

So, in a single instance of training or validation, this discrepancy can be measured with a loss function denoted as Loss(y-ŷ).

For all the data points, it can be summed as:

$$\sum_{i=1}^{N} Loss\left(y_i - \hat{y}_i\right)$$

where N is the total number of data points in the training or validation sets.

So, for instance, using stochastic gradient descent or any of its variants, the whole point of training is to minimize the loss function as shown below:

$$\theta := \theta - \partial \nabla \theta \text{Loss(y-}\hat{y}\text{)}$$

where θ is the trainable parameter, ∂ is the adjustable learning rate.

Fig. 7 shows that as the training time increases, the calculated training and validation accuracy results increase, meaning the model is performing better. At epochs 2.9 and 5.9, the results for both validation and training seem to converge, and after that they diverge meaning that the model is no longer improving.

Table 1 shows a more nuanced performance of the model other than just validation performance, it highlights precision, recall, and F1-Score of domain-specific models.

Table 1: Performance metrics reports

Class/Label	Precision	Recall	F1-Score	Support
0	0.98	0.75	0.85	55
1	0.83	0.99	0.90	70
Accuracy			0.88	125
Macro Avg	0.90	0.87	0.87	125
Weighted Avg	0.90	0.88	0.88	125

4.3 Comparisons of Findings from Different Domain Specific LLM

As explained earlier, in this second approach, the following domain-specific and Distilled BERT models were used for fine-tuning.

The different binary validation accuracy results are shown in the Table 2 listed below:

Among the domain-specific counterparts, it is obvious that both Distil-Bert and Code-BERT stood out in terms of performance.

We also observed that the difference between their training and validation accuracies, not more than 3%, was drastically reduced with epochs of 8–12. This means there is little or no overfitting between the two BERT versions. This finding proves the rationale behind the creation of Distil-BERT by the authors [8]. That is, the reduction in parameters retains about 95% of the performance of the BERT base model. In other words, quick mathematics will reveal that 95% of the validation result, which is 92%, listed above for BERT-Base uncased, is 87.4 which is remarkably close to 88% which happens to be the observed validation result for the Distil-BERT.

Table 2: Comparison of domain-specific LLM

SN	Distilled/domain-specific BERT	Validation performance (%)
1	SO-BERT	55%–58%
2	Sentence-BERT mini (All Mini)	75%
3	Code-BERT	88%
4	Distil-BERT-uncased	88%

Note: This table summarizes the experimental results of domain-specific LLM. The result is based on validation/evaluation accuracies. The first column specifies each LLM while the right column specifies the performance in percentage.

Also as confirmed by Sahn et al. [8], Distil-BERT retains about 97% of the original base counterpart's capabilities while being faster by 60%, since the total parameters of the BERT base were reduced by 40%. We noticed during the training process that it took less time. The reason Distil-BERT performed lower is because of the reduced parameters and as reported by the aforementioned developers, it retains 95%, not 100%, performance of the original Base-BERT.

4.4 General Comparisons of all LLMs in the Two Approaches

Table 3: General comparison of all LLMs

SN	Distilled/domain-specific BERT	Validation performance (%)
1	SO-BERT	55%–58%
2	Sentence-bert mini (All Mini)	75%
3	Code-BERT	88%
4	Distil-BERT-uncased	88%
5*	BERT-base-uncased	93%

Note: This Table 3 compares all the BERT LLM irrespective of whether it is base, distilled, or domain-specific. Again, the first column specifies each LLM, while the second one includes the validation/evaluation performance in percentage. * As already reported, although it tops all, this base LLM has, from the experimental result, high variance with a difference of up to 6%–7% between the training and validation accuracies.

4.5 The Best Models

The solution findings reveal that apart from the uncased Bert-Base model which gave a 93% validation score, either the Code-BERT or Distil-BERT are top performers for, domain-specific models, and therefore we can trust them as regards Classification of Functional or Non-Functional Software Requirements. However, we will advise caution, and recommend further training, and further observations before one can be confident in this. We mean the target variables of the datasets used are not only small but are also quite imbalanced: 59 *vs*. 41.

4.6 Comparisons with Similar Experiments from Other Authors

Researchers Alhoshan et al. [17] reported that some authors who used the same PROMISE datasets, but different algorithms, achieved recall and precisions in the range of 79 to 95. However, since the results of our experiment are comparatively better than 79 and remarkably close to 95, it

means that using LLM in RE, specifically NFR, is worth exploring and there is a lot of potential with this technique.

Table 4: Comparisons with similar experiments from others

SN	Performance (%)	Source/Authors
1	79–95	Similar authors reported by Alhoshan et al. [17]
2	60–80	Alhoshan et al. [17]
3	92	Budake et al. [19]
4	71–87	Subahi [7]
5	88–93	Our approach

Note: This Table 4 compares our findings with the performances of the related NFR-FR classifications carried out or reported by other authors.

Even if 95% seems to be better than the current result of BERT LLM, we believe that if some further hyperparameter tuning or the epoch and training time were increased, the result would beat this record. We intentionally did little tuning to save computing time and resources.

In any case, it is also a popular belief, among data scientists, that some conventional machine learning algorithms, such as Naïve Bayes, SVM, and Decision Trees, can outperform deep learning ones, especially in some tasks, typically involving tabular or highly structured data, or when the datasets are insufficient as is the case here.

Interestingly, the finding of this work agrees with the findings of Alhoshan et al. [17] that generic LLM outperforms domain-specific ones.

4.7 Validity Threats

For this experiment, the main validity threats include:

4.7.1 Insufficient and Imbalance Dataset

As has been stated, the distribution of the NFR *vs.* FR in the PROMISE dataset [16] is quite skewed (that is 59% *vs*. 41%, respectively). We could have balanced it ourselves using the conventional 50-50 sampling technique, but because the dataset was insufficient, we did not want to. Hence, this could be a validity threat, even if the proportional difference in the distribution is not very wide. As regards, the dataset being insufficient, this explains one of the rationales why we adopted LLM.

4.7.2 LLM Inherent Bias and Hallucination

This could also be a threat to the validity of the result. We countered this by fine-tuning various LLMs (such as Distil-BERT, Base-BERT, and Code-BERT) as already seen to get a sense of the overall performance. This could also be improved in the future iteration of this project by adopting optimization techniques such as Retrieval Augmented Generation (RAG), Vector Database [38], and Prompt Tuning, among others.

4.7.3 BERT Overfitting Tendency

There is a general belief, among practitioners, that BERT LLM tends to overfit. Once again, we used different fine-tuned and distilled versions of BERT to prevent this.

5 Summary, Conclusions, and Recommendations

5.1 Summary of Background and Problem

Truly, Artificial Intelligence, and NLP in particular, is at the forefront of many innovations in today's World [1]. This can be seen in the almost pervasive presence and capability of Large Language Models (LLMs) such as ChatGPT and Agentic AI tools such as AutoGen [4] and MetaGPT [5].

It is also obvious that software development is included [2]. Microsoft Copilot, Meta's Code Llama, and even ChatGPT have demonstrated their abilities to generate code and assist developers in their tasks [3]. Sadly, SDLC processes [39,40], Requirement Gathering in particular, have traditionally, and are still, being done manually by most corporations. This has always led to tedious, error-prone, and inefficient processes. This has been the case despite this stage's crucial nature [6,7] and the fact that it is considered a bridge between the design and implementation phases [41].

5.2 Summary of Research Objectives

In this work, the specific objectives addressed were: In the first place, to use one of the latest approaches in NLP, particularly BERT LLM, to build an efficient and accurate Requirement Engineering Classifier from natural language user requirements that can classify functional (FR) and non-functional requirements (NFR) [42].

5.3 Answering Research Questions and Objectives

Since our experimental findings, using Bert-Base in particular, outperformed comparably even on little fine-tuning and reduced training times, with those of the authors already cited above, particularly by Alhoshan et al. [17], Subahi [7] and Budake et al. [19], it is reasonable to adopt this LLM approach since this experimental result confirms that there is a potential in using transformer-based LLM in RE-related tasks.

Incidentally, our experiments also confirmed the report, by the developers of Distil-BERT, that it retains about 95% of the capability of the base BERT counterpart that it was distilled from, even though the parameters were reduced by 40%. We also noted that the finding of this work agrees with the findings of researchers such as Alhoshan et al. [17] that generic LLM outperforms domain-specific ones.

5.4 Specific Limitations, Recommendations for Future Works

Among all the challenges we met, the foremost was inadequate time, followed by other resources such as the financial cost of experiments.

We have already stated that the current performance of these models could be improved, though at the cost of further Google Colab GPU processing and computing time.

There was also the constraint of inadequate datasets. Though there is availability of PURE datasets as stated here https://ieeexplore.ieee.org/document/8049173 (accessed on 03 December 2024), it demands extra time for extra tagging and labeling to adapt it to the specific tasks of this thesis.

Hence, having access to a larger RE-related dataset would be very instrumental in improving the performance of our model, including reducing the observed overfitting.

Once again, we could not prolong the training time due to the above-stated reason of inadequate time and resources.

In addition, our model currently overfits, as we have noted, which is another limitation.

Going forward, interested researchers could explore other fine-tuning techniques such as Parameter Efficient Fine-Tuning (PEFT), Knowledge Distillation just like Distil-BERT, Prompt Engineering or Prompt Tuning techniques such as Zero or Few Short Prompting techniques, etc.

Interested Scientists could equally explore other software or RE-related LLMs such as BERT4RE (Bert For Requirement Engineering), SO-Bert (Domain-specific BERT that was trained on 152 million sentences from the popular Stack Overflow developer community, and others as listed by these authors Alhosan et al. [17]. One can even explore modern Meta's (formerly Facebook) software-related LLM such as Code-Llama.

Indeed, we wanted to use the popular and specialized BERT4RE (which is specifically tailored and designed for Requirement Engineering and related tasks); but it was neither available in TensorFlow nor Hugging Face hubs. But on a third-party website. We downloaded it though from this website but unfortunately, there was no apparent way to integrate the domain-specific LLM with the existing TensorFlow framework or Hugging Face Transformer library that facilitates access to various and huge LLM models.

In view of these, we believe that given enough time, one can make this BERT4RE available for other developers to use in Hugging Face and even design additional or down-streamed LLMs for software-related NLP tasks. Hence, we recommend that anyone interested in advancing this field should explore this area. We might still find a solution to this soon.

Furthermore, other Requirement Engineering tasks were not done because of inadequate time and other resources. Such tasks include the Problem of Incompleteness (often leading to requirement creep), Ambiguity detection [43–46], Problem of Imprecise [42] and Vagueness of Requirements, and even automation of the requirement elicitation process. These are interesting recommendations for future researchers interested in this area and other SDLC phases such as the Design, Implementation, and Testing stages. We hope to take up these questions and tasks in future versions of this work. Particularly, we have explained, in the second part of this work [35], and plans to implement, in the future iteration, LLM optimization techniques such as Retrieval Augmented Generation (RAG), Reinforcement Learning from Human Feedback (RLFH), including the latest alternative-Direct Preference Optimization (DPO) [47]. Others include Vector Database [38], Prompt-Engineering, Parameter Efficient Fine-Tuning (PEFT), 4-bit Quantization, Agentic design patterns and workflow [4,5], among others.

As regards other aspects of SDLC [39,40], different areas such as Model Driven Engineering (MDE) approach, generating UML [48] models such as class diagrams, and use-case diagrams could also be taken up by any interested researcher. Also, from literature reviews, lots of scientists have already worked on these MDE areas.

In fact, as demonstrated in the preceding paragraph, we hope to keep working in these related areas, that is, not only the first phase of SDLC, soon.

Acknowledgement: I appreciate all the various lecturers that contributed to making this work a success. This includes both academic and non-academic staff.

Funding Statement: None.

Author Contributions: The authors confirm their contribution to the paper as follows: study topic conception: Pius Onobhayedo; supervisions: Pius Onobhayedo, Charles Igah; data collection: Afam

Okonkwo; analysis and interpretation of results: Afam Okonkwo; draft manuscript preparation: Afam Okonkwo. All authors reviewed the results and approved the last version of the manuscript.

Availability of Data and Materials: This statement should make clear how readers can access the data used in the study and explain why any unavailable data cannot be released. The following statement is offered for reference: Data openly available in a public repository. The data that support the findings of this study are openly available in Zenodo platform at https://zenodo.org/records/268542 (accessed on 03 December 2024).

Ethics Approval: Not applicable.

Conflicts of Interest: The authors declare no conflicts of interest to report regarding the present study.

References

[1] C. Zhang *et al.*, "Generative image AI using design sketches as input: Opportunities and challenges," in *Proc. 15th Conf. Creat. Cognit.*, New York, NY, USA, ACM, Jun. 19–21, 2023. doi: 10.1145/3591196.3596820.

[2] D. P. Wangoo, "Artificial intelligence techniques in software engineering for automated software reuse and design," in *2018 4th Int. Conf. Comput. Commun. Automat. (ICCCA)*, 2018.

[3] I. Ozkaya, "Application of large language models to software engineering tasks: Opportunities, risks, and implications," *IEEE Softw.*, vol. 40, no. 3, pp. 4–8, May–Jun. 2023. doi: 10.1109/MS.2023.3248401.

[4] W. Qingyun *et al.*, "AutoGen: Enabling next-gen LLM applications via multi-agent conversation," presented at 12th Int. Conf. Learn. Represent. (ICLR), 2024. doi: 10.48550/arXiv.2308.08155.

[5] S. Hong *et al.*, "Metagpt: Meta programming for a multi-agent collaborative framework," presented at Int. Conf. Learn. Represent. (ICLR), 16 Jan. 2024. doi: 10.48550/arXiv.2308.00352.

[6] S. Wagner *et al.*, "Status Quo in requirements engineering: A theory and a global family of surveys," *ACM Transact. Softw. Engi. Method.*, vol. 28, no. 2, pp. 1–48, 2019. doi: 10.1145/3306607.

[7] F. Subahi, "BERT-based approach for greening software requirements engineering through non-functional requirements," *IEEE Access*, vol. 11, pp. 103001–103013, 2023. doi: 10.1109/ACCESS.2023.3317798.

[8] SANH *et al.*, "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter," 2020. [Online]. Available: https://arxiv.org/pdf/1910.01108

[9] Karunagaran, "How artificial intelligence can transform software gathering process?," Faculty of Sci. and Technolo. Bournemouth, Univ. Bournemouth, UK, Dec. 2020. doi: 10.13140/RG.2.2.24711.83361.

[10] R. Sonbol, G. Rebdawi, and N. Ghneim, "The use of NLP-based text representation techniques to support requirement engineering tasks: A systematic mapping review," *IEEE Access*, vol. 10, no. 4, pp. 62811–62830, 2022. doi: 10.1109/ACCESS.2022.3182372.

[11] M. Aranda *et al.*, "Effect of requirements analyst experience on elicitation effectiveness: A family of quasi-experiments," *IEEE Trans. Software Eng.*, vol. 49, no. 4, pp. 2088–2106, 1 Apr., 2023. doi: 10.1109/TSE.2022.3210076.

[12] E. A. Abdelnabi, A. M. Maatuk, T. M. Abdelaziz, and S. M. Elakeili, "Generating UML class diagram using NLP techniques and heuristic rules," presented at 2020 20th Int. Conf. Sci. Techni. Automat. Cont. Comput. Eng. (STA), Monastir, Tunisia, 2020, pp. 277–282. doi: 10.1109/STA50679.2020.9329301.

[13] H. Ammar, W. Abdelmoez, and M. S. Hamdi, "Software engineering using artificial intelligence techniques: Current state and open problems," presented at 2012 Int. Conf. Comput. Inform. Technol., 2012.

[14] B. A. Hossain *et al.*, "Natural language based conceptual modelling frameworks: State of the art and future opportunities," *ACM Comput. Surv.*, vol. 56, no. 12, pp. 1–26, 26 Aug. 2023. doi: 10.1145/3596597.

[15] X. Hou *et al.*, "Large language models for software engineering: A systematic literature review," *ACM Trans. Software. Eng. Methodol.*, vol. 33, no. 8, Dec. 2024, Art. no. 220. doi: 10.1145/369598.

[16] J. Cleland-Huang *et al.*, "NFR" distributed by "Zenodo," Mar. 17, 2007. doi: 10.5281/zenodo.268542.

[17] W. Alhoshan *et al.*, "Zero-shot learning for requirements classification: An exploratory study," *Issue Inform. Softw. Technol.*, vol. 159, Jul. 2023, Art. no. 107202. doi: 10.48550/arXiv.2302.04723.

[18] S. Shreta and P. Santo, "Integrating AI techniques In SDLC-design phase perspective," in *Proc. Third Int. Symp. Women Comput. Inform.*, India, Aug. 2015. doi: 10.1145/2791405.2791418.

[19] R. Budake *et al.*, "Challenges and future of AI-based requirement analysis: A literature review," in *Emerging Trends in Basic and Applied Sciences*, vol. 2, Kolhapur-416004, Maharashtra, India: Bhumi Publishing India, Feb. 2023.

[20] M. Schäfer *et al.*, "An empirical evaluation of using large language models for automated unit test generation," *IEEE Trans. Software Eng.*, vol. 50, no. 1, pp. 85–105, Jan. 2024. doi: 10.1109/TSE.2023.3334955.

[21] S. Yang and H. Sahraoui, "Towards automatically extracting UML class diagrams from natural language specifications," in *Proc. ACM/IEEE 25th Int. Conf. Model Driv. Eng. Lang. Syst.*, Montreal, QC, Canada, Association for Computing Machinery, Oct. 23–28, 2022, pp. 396–403. doi: 10.1145/3550356.3561592.

[22] J. Kuchta and P. Padhiyar, "Extracting concepts from the software requirements specification using natural language processing," in *Proc. 2018 11th Int. Conf. Human Syst. Interact. (HSI)*, 2018, pp. 443–448. doi: 10.1109/HSI.2018.8431221.

[23] M. Ibrahim and R. Ahmad, "Class diagram extraction from textual requirements using natural language processing (NLP) techniques," presented at 2010 Second Int. Conf. Comput. Res. Develop., Kuala Lumpur, Malaysia, 2010, pp. 200–204. doi: 10.1109/ICCRD.2010.71.

[24] Y. L.Gamage, "Automated software architecture diagram generator using natural language processing," BSc dissertation, Dept. Computer Sci Univ. of Westminster, UK, May 2023. doi: 10.13140/RG.2.2.31866.26563.

[25] H. Herchi and W. B. Abdessalem, "From user requirements to UML class diagram," presented at 2012 Int. Conf. Comput. Relat. Knowl., 2012. https://arxiv.org/ftp/arxiv/papers/1211/1211.0713.pdf.

[26] K. A. D. Arachchi, "Oshada Kasun Kiringoda "AI based UML diagrams generator"," Ph.D. dissertation, Univ. of Colombo School of Computing, Colombo, Sri Lanka, Nov. 2021.

[27] D. K. Deeptimahanti and M. A. Babar, "An automated tool for generating UML models from natural language requirements," presented at 2009 IEEE/ACM Int. Conf. Automat. Softw. Eng., Auckland, New Zealand, 2009, pp. 680–682. doi: 10.1109/ASE.2009.48.

[28] K. J. Stol and B. Fitzgerald, "The ABC of software engineering research," *ACM Transact. Softw. Eng. Methodol. (TOSEM)*, vol. 27, no. 3 , pp. 1–51, Sep. 2018, Art. No. 11. doi: 10.1145/3241743.

[29] M. Elallaoui *et al.*, "Automatic transformation of user stories into UML use case diagrams using NLP techniques," in *8th Int. Conf. Ambient Syst., Netw. Technol.*, Jan. 2018. doi: 10.1016/j.procs.2018.04.010.

[30] D. Ameller *et al.*, "Dealing with non-functional requirements in model-driven development: A survey," *IEEE Trans. Software Eng.*, vol. 47, no. 4, pp. 818–835, 1 Apr. 2021. doi: 10.1109/TSE.2019.2904476.

[31] K. Lano, S. Kolahdouz-Rahimi, and S. Fang, "Model transformation development using automated requirements analysis, metamodel matching, and transformation by example," *ACM Transact. Softw. Eng. Methodol.*, vol. 31, no. 1, pp. 1–71, 2022. doi: 10.1145/3471907.

[32] X. Franch *et al.*, "How do practitioners perceive the relevance of requirements engineering research?," *IEEE Trans. Software Eng.*, vol. 48, no. 6, pp. 1947–1964, 1 Jun., 2022. doi: 10.1109/TSE.2020.3042747.

[33] D. Hidellaarachchi, J. Grundy, R. Hoda, I. Mueller, "The influence of human aspects on requirements engineering-related activities: Software practitioners' perspective," *ACM Trans. Softw. Eng. Methodol.*, vol. 32, no. 5, pp. 1–37, 22 Jul. 2023. doi: 10.1145/3546943.

[34] J. Devlin *et al.*, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. 2019 Conf. North American Chap. Assoc. Computati. Linguist.: Human Lang. Technol.*, Minneapolis, Minnesota, Association for Computational Linguistics, 2019, pp. 4171–4186. doi: 10.48550/arXiv.1810.0480.5.

[35] Okonkwo *et al.*, "Review of traditional SDLC process, literature and NLP techniques for requirement engineering," Feb. 2024. doi: 10.13140/RG.2.2.20705.28002.

[36] Vaswani *et al.*, "Attention is all you need," presented at 31st Conf. Neural Inform. Process. Syst. (NIPS 2017), Long Beach, CA, USA, 2017. doi: 10.48550/arXiv.1706.03762.

[37] Y. Wan *et al.*, *"Kelly is a Warm Person, Joseph is a Role Model": Gender Biases in LLM-Generated Reference Letters*. Singapore: Association for Computational Linguistics, 2023, pp. 3730–3748. https://arxiv.org/pdf/2310.09219.pdf

[38] Y. Han, C. Liu, and P. Wang, "A comprehensive survey on vector database: storage and retrieval technique, challenge," 2023. doi: 10.48550/arXiv.2310.11703.

[39] K. Ali, "A study of software development life cycle process models," *Int. J. Adv. Res. Comp. Sci.*, vol. 8, no. 1, pp. 1, Jan.–Feb. 2017. doi: 10.26483/ijarcs.v8i1.2844.

[40] M. K. Sharma, "A study of SDLC to develop well engineered software," *Int. J. Adv. Res. Comp. Sci.*, vol. 8, no. 3, Mar.–Apr. 2017. doi: 10.26483/ijarcs.v8i3.3045.

[41] S. Al-Fedaghi, "Beyond SDLC: Process modeling and documentation using thinging machines," *Int. J. Comput. Sci. Netw. Secur. (IJCSNS)*, vol. 21, no. 7, Jul. 2021. doi: 10.22937/IJCSNS.2021.21.7.23.

[42] P. A. Laplante, *Requirements Engineering for Software and Systems*, 3rd ed., Roca Raton, FL, USA: CRC Press Taylor & Francis Group, pp. 25, 2018.

[43] R. Sharma. *et al.*, "Machine learning for constituency test of coordinating conjunctions in requirements specifications," in *RAISE 2014:in Proc. 3rd Int. Workshop. Realiz. Artif. Intell. Synerg. Softw. Eng.*, Jun. 2014, pp. 25–31. doi: 10.1145/2593801.2593806.

[44] A. Yadav, A. Patel, and M. Shah, "A comprehensive review on resolving ambiguities in natural language processing," *J. AI Open*, vol. 2, no. 1, pp. 85–92, 2021. doi: 10.1016/j.aiopen.2021.05.001.

[45] P. Yap *et al.*, *Adapting BERT for Word Sense Disambiguation with Gloss Selection Objective and Example Sentences*, Association for Computational Linguistics, 2020, pp. 41–46. doi: 10.48550/arXiv.2009.11795.

[46] R. Beg, Q. Abbas, and A. Joshi, "A method to deal with the type of lexical ambiguity in a software requirement specification document," in *2008 First Int. Conf. Emerg. Trends. Eng. Technol.*, Nagpur, India, 2008, pp. 1212–1215. doi: 10.1109/ICETET.2008.160.

[47] R. Rafailov *et al.*, "Direct preference optimization: Your language model is secretly a reward model," in *37th Conf. Neural Inform. Process. Syst. (NeurIPS 2023)*, 2023. doi: 10.48550/arXiv.2305.18290.

[48] S. Nasiri *et al.*, "From user stories to UML diagrams driven by ontological and production mode," *Int. J. Adv. Comp. Sci. Appl. (IJACSA)*, vol. 12, no. 6, pp. 1, 2021. doi: 10.14569/IJACSA.2021.0120637.