ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/376950755

USING ARTIFICIAL INTELLIGENCE TECHNIQUES IN THE REQUIREMENT ENGINEERING STAGE OF TRADITIONAL SDLC PROCESS

Thesis · December 2023

DOI: 10.13140/RG.2.2.28966.91203

CITATION
1

READS
1,092

3 authors, including:

Afam Okonkwo
Pan-Atlantic University
2 PUBLICATIONS **2** CITATIONS
SEE PROFILE

Charles Igah
Pan-Atlantic University
5 PUBLICATIONS **2** CITATIONS
SEE PROFILE

All content following this page was uploaded by Afam Okonkwo on 29 December 2023.

The user has requested enhancement of the downloaded file.

USING ARTIFICIAL INTELLIGENCE TECHNIQUES IN THE REQUIREMENT ENGINEERING STAGE OF TRADITIONAL SDLC PROCESS

Afam S. Okonkwo[1], Pius Onobhayedo [2], and Charles Igah[3]

[1]Author and M.Sc. Graduate at the School of Science and Technology, Pan-Atlantic University

[2] Project Supervisor at the School of Science and Technology, Pan-Atlantic University, Lagos, Nigeria.

[3] Assistant Project Supervisor at the School of Science and Technology, Pan-Atlantic University, Lagos, Nigeria.

A part of the dissertation submitted to the Computer and Information Sciences Department , School of Science and Technology, in partial fulfillment of the requirements for the award of degree of Master of Science (Data Science) of Pan-Atlantic University, Lagos, Nigeria.

https://pau.edu.ng/

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, 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 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 the meantime, is on improving requirement engineering particularly classification of Functional and Non-functional Requirements. Transformer powered BERT 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 domain-specific version Code-Bert, respectively can be reliable in these tasks.

Keywords: NLP, Artificial Intelligence (AI), Software, Software Engineering, Requirement Gathering, SDLC, Requirement Classification, Functional and Non-Functional Requirement, Large Language Model (LLM), Bert.

I. INTRODUCTION

Artificial Intelligence is leading many innovations in our World currently (Zhang et al, 2023), and it is also making lots of in-roads into the software development life cycle, the SDLC(Wangoo,2018). This section sets the background of this work, Problem Statement and Research Objectives,

1.1 Background to the Study

Artificial Intelligence (AI) is currently being seen at the forefront of driving the 5th generation of 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 on-going with arguments for and against them, in any case, there is no doubt that AI is causing lots of disruptions in many industries. A good number of people have also pointed to its risks both 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, among others. All these serves 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, ChatGPT's code generation using Generative Pre-Trained Model (GPT), etc. Stack Overflow recently reported experiments currently being carried out in many areas and, also in Google, such as using AI for code reviews, 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 (Yang and Sahraoui. 2022). 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 Software Development Life Cycle (SDLC), which outlines steps to software development such as Requirement Gathering, Requirement Specifications, Analysis, Design, Implementation, Testing (Ali, 2017) and Integration and finally Documentation, Training and Support, Deployment and Maintenance (Sharma, 2017). Requirement Engineering is obviously at the fore-front (Abdelnabi et al, 2021), with business analysts, project managers, product or project managers, and other key stakeholders teaming up to elicit these. Unfortunately, these processes have always been tedious and error prone, thus leading to codes constantly being re-worked and changed and driving up developmental costs and times. This means that there is an absolute need for software intelligence using AI and at least automation of some of these processes for efficiency. With this background the next section gives broader and more formal problem definition and statement which will be concretized after detailed literature review in Chapter 2.

1.2 Statement of the Problem

As already noted, Requirement Engineering is the first stage of the SDLC. As one of the most crucial stages (Abdelnabi et al, 2021), it sets the proper background and foundation for all other stages of the SDLC. Consequently, this stage facilitates the proper understanding of the whole project regarding its scope and core functionality, and the manner in which it is carried out can predict the success of any software project (Karunagaran, 2020). Indeed, most failures of software projects are traceable to ineffective approaches at this stage (Sonbol et al, 2022). It is considered a bridge between the design and implementation phase of SDLC (Al-Fedaghi ,2021). As such, it is expected to define the functional and non-functional requirements of the proposed software system clearly and concisely.

Undoubtedly, human analysts and other stakeholders play a key role in the requirement elicitation process. These elicitation methods can involve interviews, prototyping, questionnaires, use cases, etc. (Karunagaran, 2020). Generated documents are expressed in human languages; which the business or domain analyst is expected to transform (Abdelnabi et al, 2021) after analysis to provide a specification called Software Requirement Specification (SRS)

Unfortunately, there are inherent challenges associated with using complex human languages, in which the original documents were generated. These include issues such as ambiguity, incompleteness and inconsistencies (Maatuk and Abdelnabi,2021). Moreover, there are often trivial and insignificant details included in the documents, apart from the fact that it could be too long, for modern agile developers, and the elicitation process could be very tiresome, tedious and error prone.

Some of the above challenges have been addressed to a large extent using tools and AI techniques such as NLP (Hossain et al, 2023). However, most of them involve typical rule-based heuristics, lexical (Sonbol et al, 2022) and semi-automatic approaches, information extraction, morphological analysis, conceptual model (Hossain et al, 2023) and other related NLP techniques.

The above approaches have substantially yielded remarkable results and improvements in solving these problems (Hossain et al,2023). 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 (LLM), to better tackle these inherent difficulties found in natural language and requirement engineering processes. The above stated problems, as stated by Ammar et al (2012), are summarized and listed thus: Problem of Incompleteness (often leading to requirement creep), Problem of Ambiguity, Problem of Imprecise and Vagueness of Requirements, Inability to identify Functional and Non-functional Requirements from requirement texts.

It suffices to state here that due to time and resource constraints; it will not be possible to tackle all these problems at once; however, we believe that this work was able to address the last issue.

1.3 Aim and Objectives of the Project Work

Aim

This project aims to improve the Requirement Engineering stage of traditional SDLC in software industries and to contribute to the work of software developers and data scientists especially in Nigeria software companies

Specific Objectives

The specific objectives of this thesis include:

1 To use the latest approach in NLP to build an efficient and accurate Requirement Engineering Classifier from natural language user requirements that can classify functional (FR) and non-functional requirements (NFR).

2. To evaluate and compare my findings with others.

II. LITERATURE REVIEW

2.1 Overview

In this chapter, a review of the related literature will be provided. These reviews would be divided into these categories: Linguistic/Grammar Rules, GUI/Diagrammatic Aided Builder Tools, Rule-Based/Parsing/ Pattern Matching Heuristic Technique 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. Authors Kuchta and Padhiyar (2018) 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 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. Their contribution helped to solve the problem of ambiguous concepts. 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 elicitations and management. According to Al-Qaoud and Ahmad (2010), 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.

2.2.3 Rule-Based/Parsing/ 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 (2023), recent studies in these areas use part of speech (POS) tagging, domain centered databases, domain ontology and entities. According to Zhou and Zhou (2004), as reported by Ahmad (2010), 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 OO (Object Oriented) classes are discovered via part of speech tagging, grammar parsing and linguistic patterns, finally refinement is done using domain ontology. Al-Qaoud and Ahmad (2010) incorporated another heuristic based approach that they called Taxonomic Class Model (TCM), and it involves several modelling rules including analysis of nouns, using structural rules of English sentence, class categories among other heuristic rules.

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 (2023), these models are good in identifying links and relationships that exist among components. Such popular ML algorithms, used, include SVM (Support Vector Machine), Tree based algorithm such as 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 Gamage (2023) reported that authors Maatuk and Abdelnabi (2021) made use of NLP in combination with heuristic rules to extract UML artifacts, with the aid of 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, Naive Bayes classifier was used by Arachi (2022) for the attributes and key terms needed to generate UML components. In other areas, advanced algorithms such as CNN and GAN (Generative Adversarial Network) were used in the generation of architectural diagrams.

2.3 Tabular Literature Review

Table 1 below gives a more specific summary of the different works by some of the above authors, among others. Similarly, it is ordered by years in ascending order. Evaluation results, my comments, and observations, regarding each contribution, are also included on the right-hand side of the table.

Table 1

Specific summary of Contribution of various authors to requirement engineering

SN	AUTHOR(S)	TECHNIQUE	REVIEW SUMMARY	MY COMMENTS
1	(Kumar and Babar, 2009)	NLP, Rule-Based	A 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: 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 glossary, eight syntactic reconstruction rules to solve the problem of incompleteness, ambiguities and other related communication gaps.	Though there was no clear mention of performance metrics, testing as well as validation. However, the unique feature is the ability to generate not only UML use-case but also analysis class models from natural language (NL) texts. Eventually, it uses the resultant use-case to produce both collaboration and design class models of UML.
2	(Ammar et al, 2012)	Theoretical work	The authors 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 were able to articulate 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.	The trio described many systems where these techniques had been implemented using tools such as Fuzzy logic, Bayesian networks, etc. They, also, gave an extensive literature review for requirement engineering. Finally, the researchers listed open problems such as disambiguating NL requirements, use of computational intelligence to solve problems of incompleteness and requirements prioritization, building knowledge-based system and ontologies

				for managing of requirements and modelling problem domains
3	(Herchi and Abdessalem ,2012)	NLP and Domain Ontology and Heuristics	These scientists implemented a similar approach as afore-mentioned Al-Qaoud and Ahmad (2010) RACE system. But in addition, they introduced XML and used a text-processing java-based *GATE* open-source framework, built by 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.	The solution performed better (recall:83%, precision 93%) in comparison with CM-Builder using case studies published in OO analysis books. However, as is the case with heuristics rules, it didn't cover extensive rules. Even though it solved the target problem, class diagram generation; it didn't, however, cover the dynamic parts of the UML artifacts
4	(Kuchta and Padhiyar, 2018)	NLP and Grammatical/Linguistic Analysis using WordNet Ontology	The writers proposed a methodology different from domain ontology and traditional statistical approaches due to their inherent weaknesses. Rather 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.	Their contributions helped to solve the problem of ambiguous concepts. Their next plan of action, as of the time of writing, was to generate class diagrams from the concepts that had been disambiguated.
5	(Karunagaran, 2020)	Theoretical Proposal	The author tried to propose 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 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.	He implemented interesting modern approaches, including Bag of Words which is not bad as it is currently in use even though it has problems of not including context, being unordered and a high dimensionality issue among others. He furthermore considered ethical concerns which are praiseworthy.
6	(Budake et al 2023)	Theoretical Literature Review	These researchers 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 quiet and modern NLP data pre-processing features such as n-grams and bag 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,2012). Lastly, following their discussions, the	We think the authors did extensive literature review and theoretical works, though their work didn't describe any system they implemented, however their theoretical proposals and suggestions can serve as foundation for future work.

			authors recommended the following should be done, by anyone interested in this area, before using NLP in requirement gathering stage of SDLC: Domain knowledge and modelling, 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.	
7	(Hossain et al, 2023).	Theoretical Work-Model Driven Engineering, Surveys	In this work, the researchers extensively surveyed the different traditional and popular NLP powered conceptual modelling frameworks. Specifically, they studied how each system was constricted, the architecture, motivations, verification capability, among others.	It was indeed an extensive survey including some of the above reviewed frameworks such as RACE, UGAR, NLOOPS, GATE, among others. They also suggested future opportunities for research.
8	(Alhoshan et al, 2023)	ML, Zero-Shot Learning, Word Embeddings and Transformer-Based LLM	Using Zero-Shot Learning method as well as Transformer-based LLM, the researchers observed that the scientific community had been focusing on using supervised learning technique, which has the inherent problems of relying exclusively on labelled data, as regards solving Requirement Engineering (RE) challenges. Unfortunately, these labelled 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 labelled data.	Their experiment proved that this novel approach is promising with an F1 score of 0.66 for the functional and non-functional classification. As regards Non-Functional Classification (NFR), the F1 is in the range of 0.72 -0.80. Personally, we find the approach encouraging, especially as regards the perennial issue of shortage of data in RE tasks.

Note: This table 1 gives a degree of detailed literature review of different contributions of various researchers in requirement engineering.

2.4 Observations from the 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*. This means this novel technique is yet to be fully and sufficiently explored. Hence, this work will be exploring this approach/methodology to contribute and advance this field.

III. METHODOLOGY

3.1 Overview

For this chapter, the sections are divided into: Brief Statement of proposed Methodology and Technical Approach, Proposed Architecture, Reason for the Proposed Architecture/Technical Approach, Description of Datasets and its Source, Brief Description of Architectural Components and Proposed Technologies, Frameworks and Libraries, Description of Algorithmic Steps, Evaluation Metrics and Ethical Considerations.

3.2 Research Design and Statement of Methodology and Proposed Technical Approach

Adopted Approach: ***Fine-Tuning of BERT (Bidirectional Encoder Representations from Transformers) LLM and Attachment of Classification Head to the Transformer based Model.***
Classification of Requirements Texts into Functional (FR) and Non- 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, also to know areas to focus initial attentions. 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 (Bidirectional Encoder Representations from Transformers), introduced by Google in a paper by Devlin et al (2018). 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. SO-Bert: Software domain-specific BERT trained on 152 million sentences from Stack Overflow, a platform for software developers. This is particularly good and trained for down-stream tasks such as Named Entity Recognition (NER).
2. Bert Mini (All Mini): A miniaturized and distilled version of BERT that is smaller in size but nevertheless has most or all the performance capability of base BERT.
3. Code-Bert: A version of BERT trained and tailored for both programming texts and language texts.
4. Distill-Bert: Another version of BERT that is lesser in size than the original BERT base model, but nevertheless retains about 95% of the original base counterpart. It is reported to have reduced the total parameters of the BERT base by 40%.

3.2.1 Proposed System Architecture

The proposed overall system architecture is shown in figure 3.0a

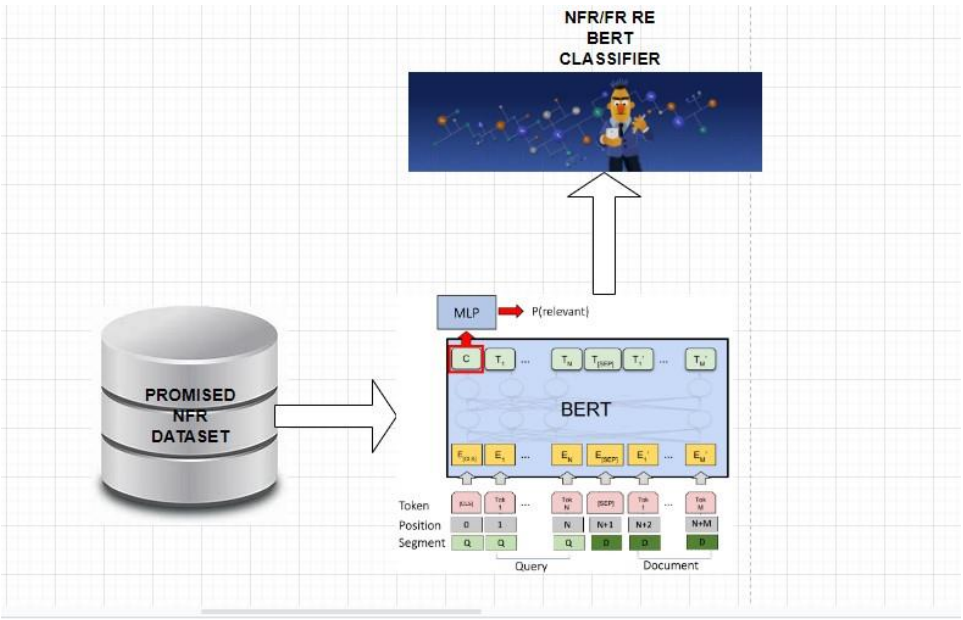


Figure 3.0a: The complete system architecture

Figure 3.0a summarizes the whole and overall system. Typically, the PROMISED NFR datasets are inputted into BERT LLM, the output is a BERT NFR/FR Classifier for Requirement Gathering.

Going forward the overall architectural components will be broken down, starting with BERT.

The proposed LLM BERT architecture is shown in figure 3.0b

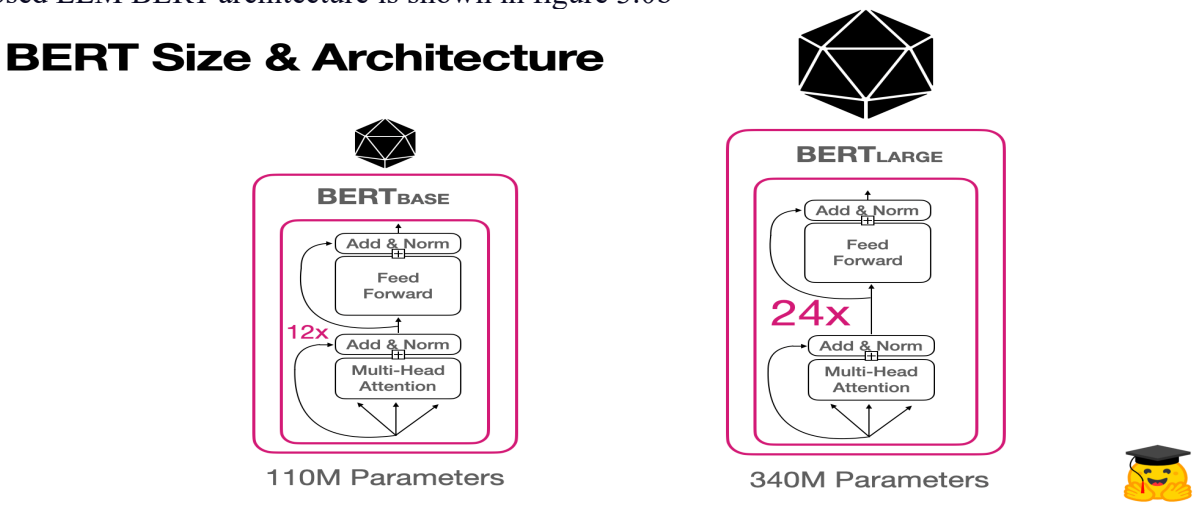


Figure 3.0b: Bert Size and corresponding Architecture

Image Credit: https://huggingface.co/blog/bert-101#3-bert-model-size--architecture

3.2.2 Reasons for the Proposed BERT Architecture and Technical Approach

The above choice was informed by the following reasons:

Why LLM? LLM was used for the following reasons:

1. LLM is a pretrained 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, carbon footprints and thus leads to a sustainable and safer climate, apart from efficiency of time and other resources during training.

Reasons for BERT:

I. BERT is reported to be particularly good in software domains (Alhoson et al, 2023)

II. It has lots of pretrained extensions capable of performing a wide array of tasks. Currently, it has at least 3 LLMs available in Hugging Face's NLP hub, that are 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 attentions. This will be dealt with in detail in the following sessions.

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 task. 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.

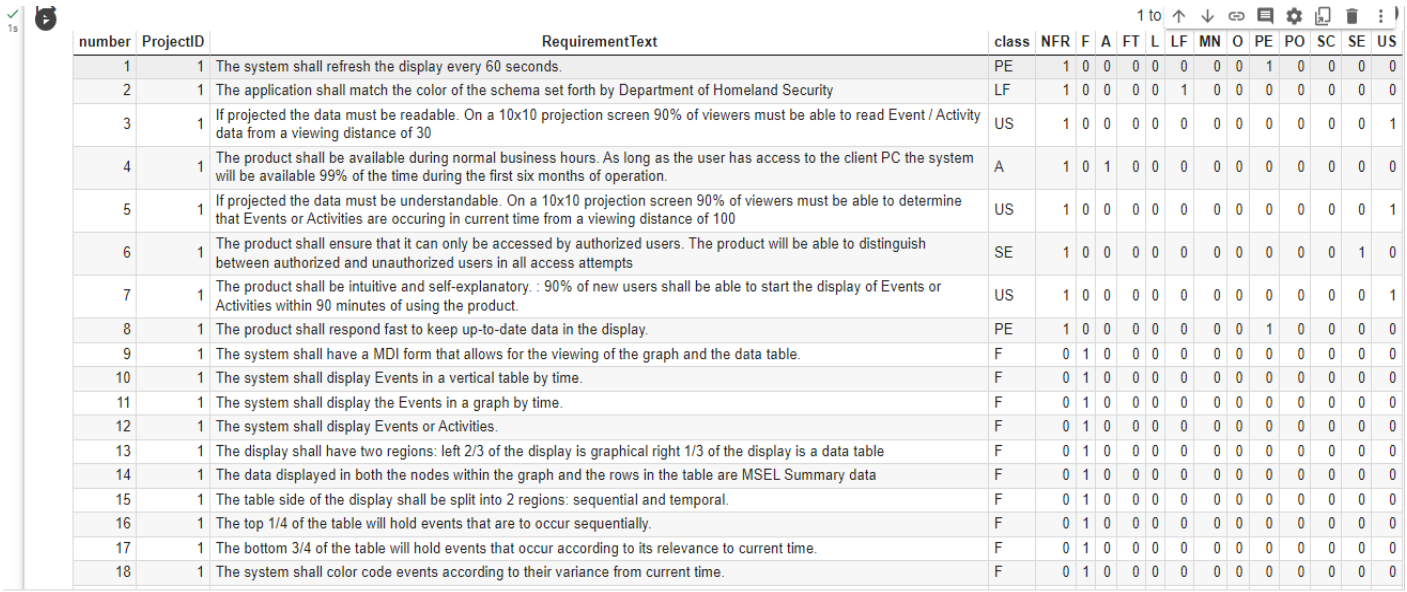
Since context, precision and ambiguity detection are particularly important in software requirement analysis, the above constituent features make BERT a desirable choice.

Reasons for Google TensorFlow and Hugging Face:

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

3.2.3 Description of dataset and its source

Since this is a software requirement task, the dataset that will be used is the PROMISE NFR dataset, available here https://zenodo.org/records/268542 , a popular software requirement dataset widely used in Requirement Engineering by researchers. The dataset, introduced by Cleland-Huang et al (2007), 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 totalled 67, Fault Tolerance (FT) has just 10, and Portability denoted as PO is the least with just only 1 requirement belonging to its class. Figure 3.1a shows the first 10 rows of the dataset as displayed by the Data table of Google Colab. The excel screen shot is also shown.



1 to

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 3.1a: A View of Promise Datasets from Google Colab's Data Table API

As can be seen, the class label represents different classes of the NFR already listed above. Our target column, 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.

3.2.4 Brief Description of Architectural Components, Frameworks and Libraries

In this section, there will be a brief introduction and description of the main proposed technologies used in this experimental design. They include Transformers, BERT, TensorFlow, Keras and Hugging Face.

Transformers:

Introduced in 2020 paper called *Attention is all you need* by Vaswani et al, Transformers have popularized NLP applications lately. It is simply a natural language processing architecture that uses *Attention mechanism* to boost the performance of NLP applications. With Attention, it can dispense with previous recurrent and convolutional networks as reported by its authors. It is better than Convolutional Neural Network (CNN) because it is parallelizable due to its popular multi-head attentions. At the same time, it outperforms RNN, and its LSTM, due to its long-range dependencies. This innovative NLP architecture has powered many text-based applications including LLMs such as Google's BERT and OpenAI's ChatGPT's. Transformer performs very well in texts that are sequence based by default such as Machine Translation and Sequence to Sequence Model. Figure 3.2a shows a basic Transformer while Figure 3.2b shows more holistic architecture.

Figure 3.2a: A Basic and abstracted Transformer Architecture
Image Credit & Source: https://towardsdatascience.com/transformers-explained-visually-part-1-overview-of-functionality-95a6dd460452

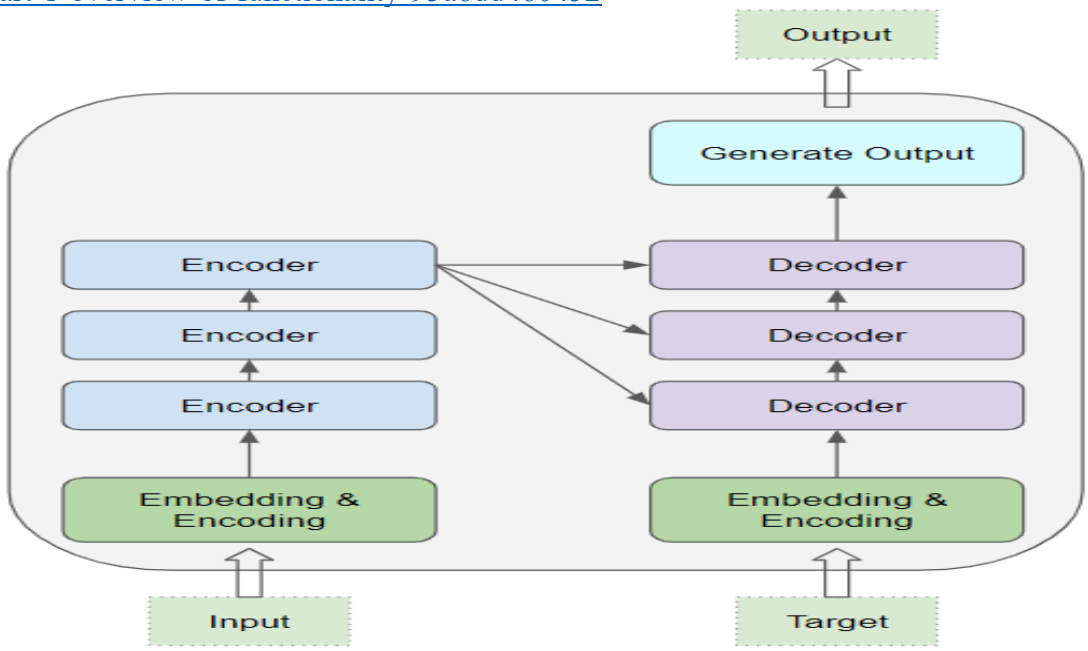


Figure 3.2b: A detailed and explored Transformer Architecture
Image Credit & Source: https://towardsdatascience.com/transformers-explained-visually-part-1-overview-of-functionality-95a6dd460452

As shown in the last figure, at its core, Transformer is made up of a stack of encoders and decoders, positional encodings, and embeddings, as well as input and output sequences. All the encoders are similar. Likewise, all the decoders are similar. Note that the Encoder contains the crucial *Self Attention Layer* and Feed Forward Neural Network Layer. In like manner, Decoder contains Feed-Forward and additional Encoder- Decoder Attention Layer. Finally, each encoder or decoder has its own set of weights.
Attention is at the core of its impressive performance. It helps the model to focus on other words that are closely related to the current word being processed in the sequence.
Example.: *The boy is wearing a black shirt*.
Here, *Shirt* is more closely related to 'black' and 'wearing' than is black related to the boy. Hence, transformer architecture uses self-attention by relating every word in the input sequence to every other word. It can understand context very well. Example:

- The *cat* drank the milk because **it** was hungry.
- The cat drank the *milk* because **it** was sweet.

In the second sentence, the attention would make the model understand that "it" refers to milk and not cat.
Finally, the transformer uses multiple attention scores for each word to handle more fine-grained intent and semantics.

BERT:

Bidirectional Encoder Representations from Transformers (BERT), as stated earlier, was developed in Google by Devlin et al in a 2018 paper titled *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. As a language representation model, it can pre-train a deep bidirectional representation from semi-supervised and unlabeled texts, looking at both left and right contexts. Concretely, BERT is built on top of earlier powerful NLP techniques: ELMO, ULM-FiT, OpenAI Transformer and Transformer in general, it passed through semi-supervised training on enormous amounts of corpora including thousands of books, Wikipedia, etc., as well as supervised training on labelled dataset. Figure 3.4 shows two steps followed when building BERT LLM:

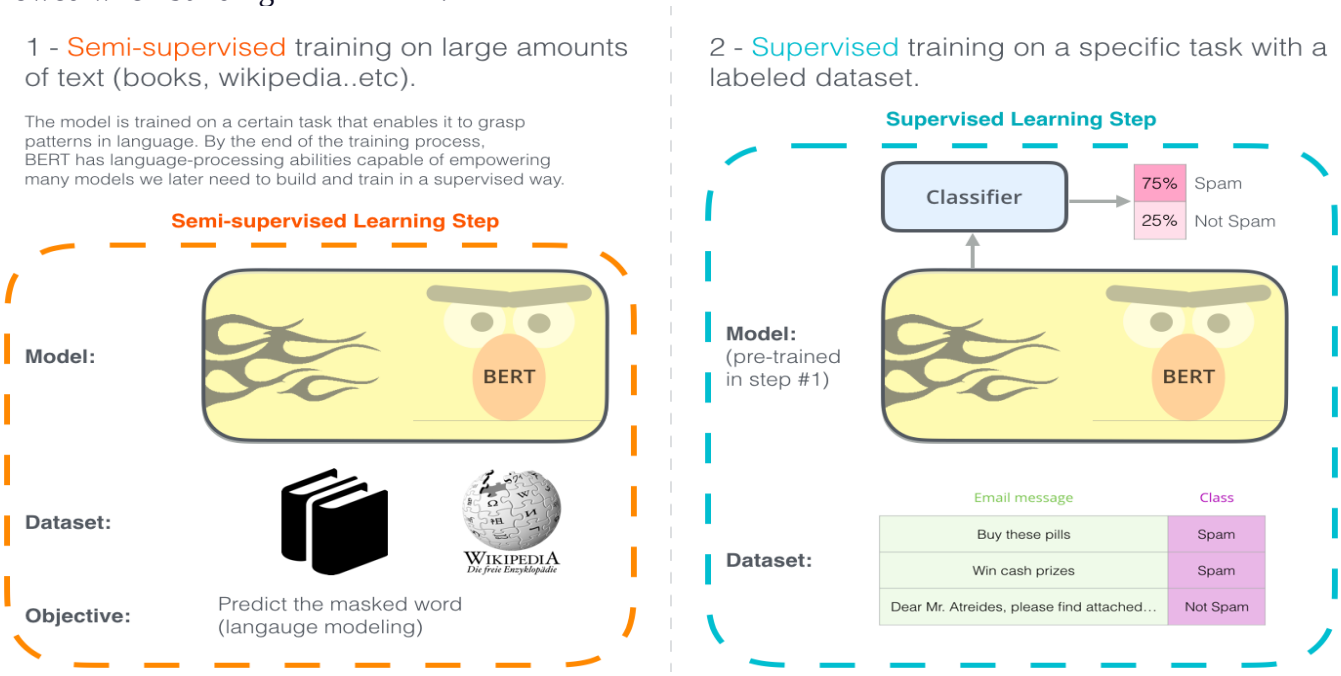


Figure 3.4: Two Steps of BERT Training
Image Credit & Source: https://jalammar.github.io/illustrated-bert/

Normally, one needs to train only the classifier, that is adding a classification head to the BERT base. This is known as Fine-Tuning, and it is exactly what this experiment intends to achieve in this chapter. This is also called Downstream Task.
BERT can also be used, apart from Fine-Tuning, to build another Language or Text Embeddings model.

TensorFlow: Also Developed by Google, this free and open source-software python framework is used for deep learning. one can use pre-trained models or alternatively train a new one. It is immensely helpful not only for training models but also for preparing data, deploying models, and implementing Machine Learning Operations (MLOP).

Keras: A simplified, high-level, and deep learning Application Programming Interface (API), it is used to implement neural networks, written in python programming language and is good for rapid prototyping and experimentation, it provides user friendly interface and is beginner friendly.

3.2.5 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: This involves setting up GPUs runtimes, TensorFlow and its auxiliary models and libraries such as Data API, TensorFlow Hub, etc.

2. Task 2: Load the PROMISE NFR Dataset and carry out some Exploratory Data Analysis (EDA).

3. Task 3-Create tf.data.Datasets for Training and Evaluation: This involves splitting the datasets into traditional train and validation pairs and since we are working with TensorFlow, this task requires converting the pre-processed and partitioned data into tensors for easier processing by TensorFlow.

4. Task 4- Download a Pre-trained BERT Model from TensorFlow Hub: In this step, we will download a pre-trained BERT based model called *bert-en-uncased* from TensorFlow hub and then set the parameter *trainable* to be true, since we want to fine-tune it.

5. Task 5- Tokenize and Pre-process Text for BERT: Typically, BERT accepts three types of input format:

a. Input word ids: These represent the base vocabulary ids used during pre-training of BERT. Of course, there is a fixed length represented by arbitrary chosen max length- this is typically the same size as the largest sentence in the dataset.

b. Input mask: This contains 0 or 1 flag which verifies if the token in the input word id is zero padded or not. If it contains a valid token, input mask will have 1, else it will contain zero.

c. segment ids: In our case, we do not need this feature since we are performing fine-tuning of the transformer-based model by attaching classification head. It is required by BERT because of its second task, as explained earlier, called Next Sentence Prediction (NSP). We will pass only one sentence here, no need for a second sentence in this case.

6. Task 6- Wrap a Python Function into a TensorFlow op for Eager Execution:

Ordinarily, the dataset map function runs in graph mode. Consequently, graph tensors do not have a value. Moreover, one can only use TensorFlow Ops and functions in graph mode. However, we need to map each element in the datasets into the above required BERT data formats. Thus, we need a Python function wrapper for eager execution that will make this possible.

7. Task 7: Create a TensorFlow Input Pipeline with tf.data: For this task, we will set up an input pipeline using TensorFlow tf.data. This is especially important for the sake of efficiency of data streaming during training. That is, to fast-track Input/Output (I/O) operations and avoid bottlenecks that are typically involved in reading from disk. Also, we will specify the batch size, shuffling and other hyper parameters for optimization of the entire process.
8. Task 8-Add a Classification Head to the BERT hub.KerasLayer: At this point, we will use the popular Keras Input function to set up the three input types accepted by BERT. We will then pass the three parameters to the already downloaded Bert's uncased base model for processing. Next, we will get the resultant pooled outputs from the base BERT model. After applying optimization techniques, such as drop out, we will then attach it to Keras' One-layer Dense Model as a Classification Head. Finally, we shall create a typical trainable Keras Model, passing the above 3 inputs already formatted for Keras.
9. Task 9- Fine-Tune BERT for NFR Classification: In this penultimate step, we will simply compile our model after passing the necessary parameters such as optimizer type, loss function as well as evaluation metrics. Lastly, we shall fit the model after passing, as parameters, the required training, validation data and epoch value.

10. Task 10: Evaluate the BERT NFR Classification Model: Here, we shall evaluate the performance of the newly fine-tuned BERT model.

The second algorithmic approach, as Illustrated in figure 3.7, consists of smaller set of tasks and involves using Hugging Face Transformer and related libraries

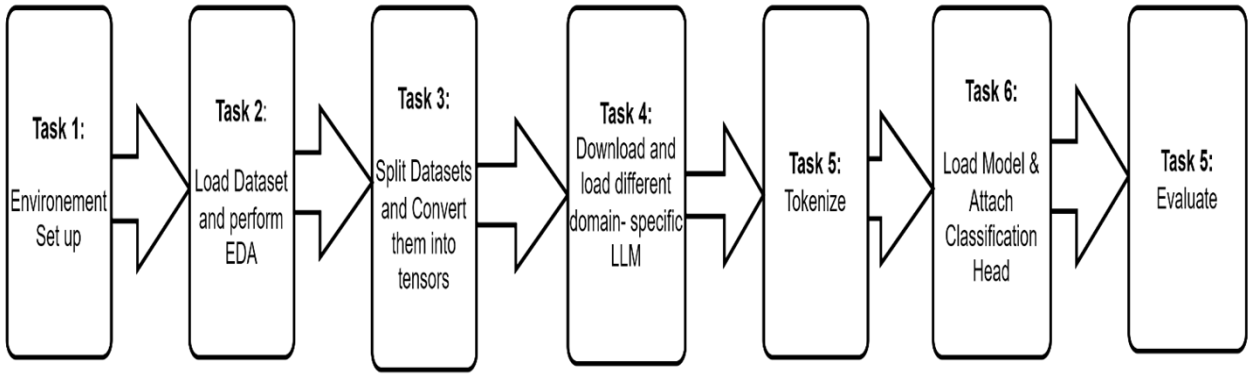


Figure 3.7: An illustration of the second algorithmic approach.

They are explained in detail as follows:

1. **Task 1- Setting up TensorFlow, Google Colab Runtime and Hugging Face Libraries:** Like the above task 1, this also involves installing and importing Hugging Face Transformer and its related libraries and models.
2. **Task 2-Load the PROMISE NFR dataset and carry out some Exploratory Data Analysis (EDA):** This step is the same as task 2 of approach 1.
3. **Task 3-Create tf.data.Datasets for Training and Evaluation:** The same as approach1.
4. **Task 4-Download and Load Different Versions of Pre-trained BERT Models from Hugging Face Repo:** In this step, we shall download a 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**: For each loaded and downloaded BERT specific-domain LLM, we shall get its tokenizer, use it to tokenize the dataset and once again prepare it for the

formats accepted by these different LLMs. These formats are input ids, and attention masks.

6. **Task 6-Load Model and Attach Classification Head:** After tokenization, for each loaded and downloaded BERT specific-domain LLM, we shall load the model. Finally, we will create a model from transformer's *TFAutoModelSequenceClassification* using its *from pretrained* function and passing the model title as parameter. Then, we will compile our model after passing the necessary parameters such as optimizer type, loss function and evaluation metrics. Lastly, we shall fit our model having passed the required training and validation data including an epoch parameter.
7. **Task 7- Evaluate the BERT NFR Classification Model:** Here, we shall evaluate the performance of the newly fine-tuned BERT model. We intend to evaluate and compare the performance of different versions of BERT used above.

3.2.6 Evaluation Metrics

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 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 fine-tuned BERT model that does binary classification.
3. We shall make use of Adam as 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 training process, ***EarlyStopping,*** this stops the training process when the model performance is no longer improving. This prevents waste of resources and computer time.

3.3 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 subsequent experiment. Also, the dataset is freely available and has been used extensively by the research community as confirmed by Alhoshan et al (2023). 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 racial 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: Wan et al (2023) carried out a study that revealed gender bias in LLM generated reference letters. There are also popular cases of hallucination bias.

IV. 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. Figures 4.1 and 4.2 both show experimental outputs.

```
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 4.1: TensorFlow-Keras Model Summary of Trainable Parameters and Model Size

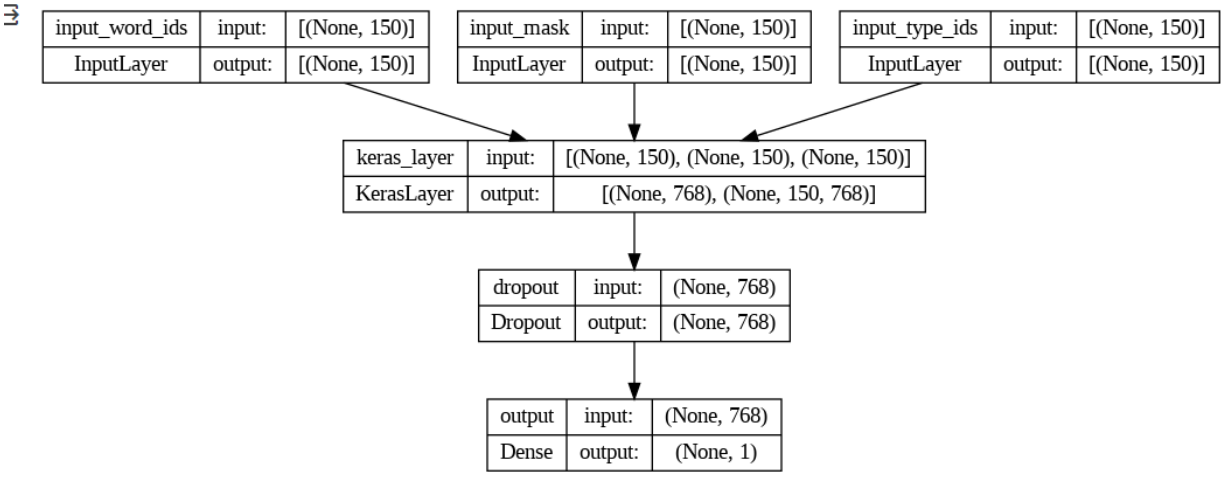


Figure 4.2: Keras Visual Plot of Trainable Parameters

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

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

This is obviously the case of overfitting, however there is no underfitting, this is because in comparisons with performances of other algorithms, such as SVM, Naïve Bayes, etc. reported by Alhoshan et al (2023), this model performs better, however it suffers from overfitting. BERT, and in general many large deep learning models, are prone to overfitting. Nevertheless, we believe that further tunning or hyper parameter tuning can reduce the overfitting. Especially if we increase the epoch value which is between 4-5 in this case.

4.2 Observations from the Second Algorithmic Approach: Domain Specific LM via Hugging Face Transformer Hub:

Please find figures ***4.3 and 4.4*** plots of respective training-validation loss and training-validation accuracy for Code-BERT model. It is plotted against epoch.

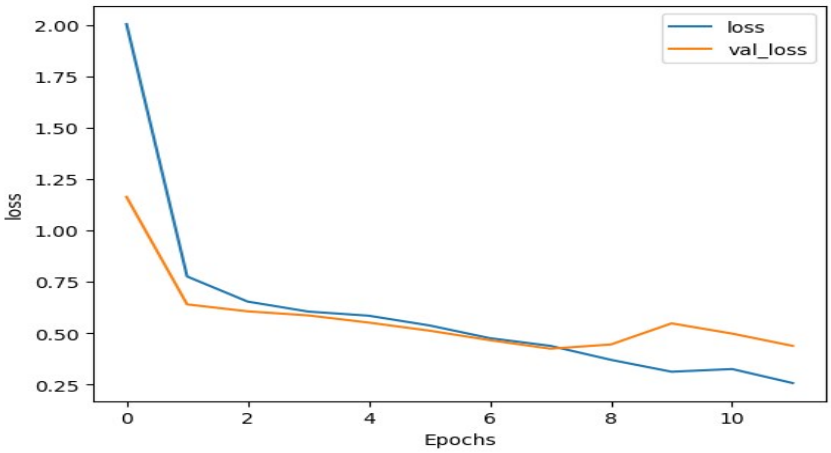


Figure 4.3: Training-Validation Loss plotted against Epoch

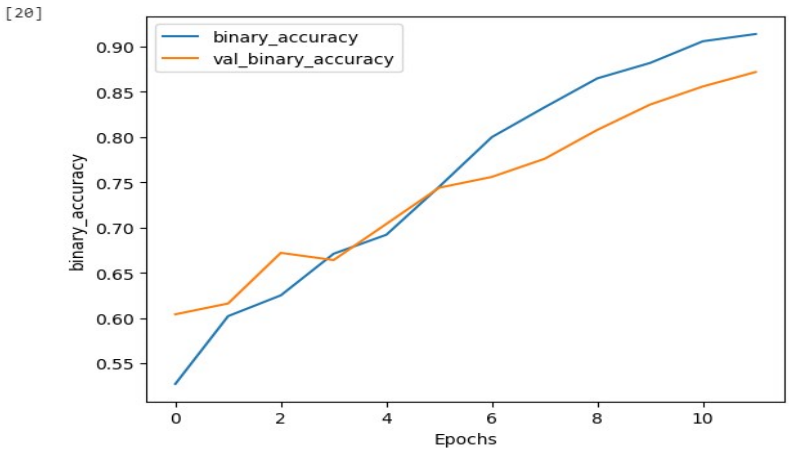


Figure 4.4: Training-Validation Accuracy plotted against Epoch

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 3 listed below:

Table 3
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.

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 epoch 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. That is, that the reduction in parameters retains about 95% of the performance of BERT base model. In other words, a 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% that happens to be the observed validation result for the Distil-Bert

4.4 General Comparisons of all LLMs in the two approaches

Table 4

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	Distill-Bert-Uncased	88%
5*	Bert-Base-Uncased	93%

Note: This table 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 uncased Bert-base model which gave 93% validation score, either of 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, 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

Table 5

Comparisons with Similar Experiments from Others

SN	SOURCE	PERFORMANCE (%)
1	Similar authors reported by Alhoshan et al (2023)	79-95
2	Alhoshan et al (2023)	60-80
3	Budake et al (2023)	92
4	Our Approach	88-93

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

Alhoshan et al (2023) reported that some authors who used the same PROMISE datasets, but different algorithms, achieved recall and precisions in the range of 79 to 95.

Though 95% seems to be better than the current result of Bert LLM, we believe that if the epoch and training time were increased, the result would beat this record. We intentionally reduced these values to save computing time and resources.

In any case, it is also a popular believe, among data scientists, that some conventional machine learning algorithms, such as Naïve Bayes, SVM, 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 ,2023) that generic LLM outperforms domain-specific ones.

V. Summary, Conclusions and Recommendations

5.1 Summary of Background and Problem

Truly, Artificial Intelligence, and NLP in particular, is at the forefront of many innovative innovations in today's World (Zhang et al, 2023). This can be seen in the almost pervasive presence and capability of Large Language Models (LLMs) such as ChatGPT. It is also obvious that software development is not left out (Wangoo,2018). Microsoft Copilot, Meta's Code Llama and even ChatGPT have demonstrated their abilities to generate code as well as assist developers in their tasks. Sadly, SDLC processes, Requirement Gathering in particular, have traditionally, and are still, been done manually by most corporations. This has always led to tedious, error prone and inefficient processes.

5.2 Summary of Research Objectives

In this work, the specific objectives addressed were: In the first place, to use the latest approach 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). Secondly, to evaluate the performance of the approach.

5.3 Answering Research Questions and Objectives

Since our experiments, BERT-Base particularly outperformed comparably, even on little fine-tuning and reduced training times, with those of the authors already cited, it is reasonable to adopt this LLM approach.

Incidentally, our experiments also confirmed the report, by the developers of Distill-BERT, that it retains about 95% of the capability of 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 (2023) that generic LLM outperforms domain-specific ones.

5.4 Specific Limitations, Recommendations for Future Works

Among all the challenges we met, the foremost is 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 challenge of inadequate datasets. Though there is availability of PURE datasets as stated here https://ieeexplore.ieee.org/document/8049173 ,it however demands extra time for extra tagging and labeling to adapt it to the specific tasks of this thesis.

More importantly, 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 existing TensorFlow framework or Hugging Face Transformer library that facilitate access to various and huge LLM models.

In view of these, we believe that giving 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.

In addition, different areas such as Model Driven Engineering (MDE) approach, generating UML models such as class diagrams, use-case diagrams could be taken up by any interested researcher. Also, from literature reviews, lots of scientists have already worked on these MDE areas.

In conclusion, other Requirement Engineering tasks were not done for inadequate time and other resources. Such tasks include Problem of Incompleteness (often leading to requirement creep), Ambiguity detection, Problem of Imprecise and Vagueness of Requirements and even automation of requirement elicitation process. These are interesting recommendations for future researchers interested in this area and other SDLC phases such as Design, Implementations and Testing stages. We hope to take up these questions and tasks in the future parts of this work.

Relatedly, as regards implementation phase of SDLC, we are currently finishing the process of fine-tuning a mini version of Meta's (formerly Facebook) latest LLM called Code Llama. Code LIama is a down-streamed version of the popular Llama 2 LLM. It can generate code and output natural languages from code.

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.

VI. REFERENCES

Cleland-Huang J. et al NFR (2007). Dataset URL https://doi.org/10.5281/zenodo.268542

Kumar, D. D. and Babar, M. A. (2009). An automated tool for generating UML models from natural language requirements. In Proceedings of the 2009 24th IEEE/ACM international conference on Automated Software Engineering. IEEE Computer Society, 680ś682. Available from https://doi.org/10.1109/ASE.2009.48

Al-Qaoud, M.I. and Ahmad, R. (2010). Class Diagram Extraction from Textual Requirements Using Natural Language Processing (NLP) Techniques. In 2010 Second International Conference on Computer Research and Development. IEEE, 200ś204. Available from https://doi.org/10.1109/ICCRD.2010.71

Ammar H., Abdelmoez W., Hamdi M.S. (2012) Software Engineering Using Artificial Intelligence Techniques: Current State and Open Problems. West Virginia University, Ahmed Bin Mohammed Military College, Arab Academy of Science, Technology and Maritime Transport.

Herchi H., Abdessalem W.B. (2012). From user requirements to UML class diagram, Department of computer science, High Institute of Management of Tunis, Tunisia https://arxiv.org/ftp/arxiv/papers/1211/1211.0713.pdf

Shreta, S., Santo P. (2015). Integrating AI Techniques In SDLC-Design Phase Perspective, Jagannath University and Ministry of Communications, Govt of India, India.

Ali, K. (2017). A Study of Software Development Life Cycle Process Models. MS Computer Science Department of Computer Science, Lahore Leads University Main Campus, Kalma Chouk Lahore, Pakistan. International Journal of Advanced Research in Computer Science, Volume 8, No. 1, Jan-Feb 2017. ISSN No. 0976-5697. http://www.ijarcs.info/

Sharma, M.K. (2017). A study of SDLC to develop well engineered software. Ph.D. Research Scholar (GNA University) Head, Sr. Assistant Professor Post Graduate Department of Computer Science Jagdish Chandra D.A.V. College, Dasuya, Punjab, India. International Journal of Advanced Research in Computer Science, Volume 8, No. 3, March – April 2017. ISSN No. 0976-5697. http://www.ijarcs.info/

Kuchta, J. and Padhiyar, P. (2018). Extracting concepts from the software requirements specification using natural language processing. Proceedings - 2018 11th International Conference on Human System Interaction, HSI 2018, 443–448. Available from https://doi.org/10.1109/HSI.2018.8431221
[Accessed 4 July 2023]

Wangoo, D. P. (2018). Artificial Intelligence Techniques in Software Engineering for Automated Software Reuse and Design. Ph.D. Research Scholar, Department of CSE & IT Amity School of Engineering and Technology Amity University, Noida, Uttar Pradesh, India Proceedings - 2018 4th International Conference on Computing Communication and Automation (ICCCA)
[Accessed 15th April, 2023] at 14:56:35 UTC from IEEE Xplore

Laplante P. A. (2018). Requirements Engineering for Software and Systems Pages 25, CRC Press Taylor & Francis Group, NW, Suite, Roca Raton, Florida, USA.

Elallaoui, M. et al (2018). Automatic Transformation of User Stories into UML Use Case Diagrams using NLP Techniques, University Ibn Tofail, Kenitra; Mohammed V University in Rabat, Morocco.

Karunagaran, A. (2020). How Artificial Intelligence Can Transform Software Gathering Process? Faculty of Science and Technology Bournemouth University Bournemouth, United Kingdom
Al-Fedaghi, S. (2021) Beyond SDLC: Process Modeling and Documentation Using Thinging Machines. Computer Engineering Department Kuwait University, Kuwait. IJCSNS International Journal of Computer Science and Network Security, VOL.21 No.7, July 2021 https://arxiv.org/abs/2107.13633 https://doi.org/10.22937/IJCSNS.2021.21.7.23

Abdelnabi, E.A, et al. (2021). Generating UML Use Case and Activity Diagrams Using NLP Techniques and Heuristics Rules, Faculty of Information Technology, Benghazi University, Libya.
Nasiri, S. et al (2021). From User Stories to UML Diagrams Driven by Ontological and Production Mode. LMMI Laboratory of ENSAM, Moulay Ismail University ISIC Research Team of ESTM Meknes, Morocco. International Journal of Advanced Computer Science and Applications (IJACSA), Vol. 12, No. 6, 2021. Available from http://dx.doi.org/10.14569/IJACSA.2021.0120637
Sonbol R., et al (2022). The Use of NLP-Based Text Representation Techniques to Support Requirement Engineering Tasks: A Systematic Mapping Review
Department of Informatics, Higher Institute for Applied Sciences and Technology (HIAST), Faculty of Information and Communication Technology, Arab International University (AIU), Damascus, Syria. IEEE Last Accessed at https://doi.org/10.1109/ACCESS.2022.3182372. https://arxiv.org/abs/2206.00421
Yang S. and Sahraoui H. (2022). Towards Automatically Extracting UML Class Diagrams from Natural Language Specifications. Universite De Montreal, Montreal, Canada. In ACM/IEEE 25th International Conference on Model Driven Engineering Languages and Systems (MODELS '22 Companion), October 23–28, 2022, Montreal, QC, Canada. ACM, New York, NY, USA, 8 pages https://arxiv.org/abs/2210.14441 https://doi.org/10.1145/3550356.3561592.

Budake, R. et al. (2023). Challenges And Future Of AI-Based Requirement Analysis: A Literature Review. Department of Computer Science K.H. College, Gargoti-416209, and Shivaji University, Maharashtra, India 2CSIBER, Kolhapur-416004, Maharashtra, India.

Gamage, Y. (2023). Automated Software Architecture Diagram Generator using Natural Language Processing. A Dissertation by Mr Yasitha Lalanga Gamage Submitted in partial fulfillment of the requirements for the BSc (Hons) in Computer Science degree at the University of Westminster, UK. May, 2023 http://dx.doi.org/10.13140/RG.2.2.31866.26563
Zhang, C. et al (2023). Generative Image AI Using Design Sketches as input: Opportunities and Challenges. In Creativity and Cognition (C&C '23), June 19–21, 2023, Virtual Event, USA. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3591196.3596820
Hossain, B.A et al (2023). Natural Language Based Conceptual Modelling Frameworks: State of the Art and Future Opportunities *ACM Computing Survey* Available from https://doi.org/10.1145/3596597
Sharma, R. et al (2014) Machine Learning for Constituency Test of Coordinating Conjunctions in Requirements Specifications. RAISE 2014: Proceedings of the 3rd International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering June 2014 Pages 25–31 http://dx.doi.org/10.1145/2593801.2593806

Vaswani, A. et al (2017). Attention Is All You Need. Google Brain and Google Research Team. 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA https://doi.org/10.48550/arXiv.1706.03762
Hayman, Oo. et al (2018). An Analysis of Ambiguity Detection Techniques for Software Requirements Specification (SRS). International Journal of Engineering and Technology. 7(2.29):501 Available at https://doi.org/10.14419/ijet.v7i2.29.13808

Devlin, et al (2018) BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding Available at https://doi.org/10.48550/arXiv.1810.04805 Google AI Researcher Team.

Yap, P. et al (2020). Adapting BERT for Word Sense Disambiguation with Gloss Selection Objective and Example Sentences. Available at https://doi.org/10.48550/arXiv.2009.11795

Yadav, A. et al (2021). A comprehensive review on resolving ambiguities in natural language processing. A Journal of AI Open Volume 2, 2021. Pages 85-92. Last Accessed from https://doi.org/10.1016/j.aiopen.2021.05.001

Alhoshan, W. et al (2023). Zero-Shot Learning for Requirements Classification: An Exploratory Study. Issue of Information and Software Technology Volume 159, July 2023, 107202. Available at https://doi.org/10.48550/arXiv.2302.04723

Wan, Y. et al (2023). “Kelly is a Warm Person; Joseph is a Role Model”: Gender Biases in LLM-Generated Reference Letters. Available at https://arxiv.org/pdf/2310.09219.pdf

View publication stats