IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 25 August 2023, accepted 18 September 2023, date of publication 20 September 2023,
date of current version 26 September 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3317798

RESEARCH ARTICLE

BERT-Based Approach for Greening Software Requirements Engineering Through Non-Functional Requirements

AHMAD F. SUBAHI
Department of Computer Science, University College of Al Jamoum, Umm Al-Qura University, Makkah 21421, Saudi Arabia
e-mail: afsubahi@uqu.edu.sa

ABSTRACT The incorporation of sustainability principles during the requirements engineering phase of the development life cycle constitutes greening software requirements. This incorporation can have a variety of effects on the software design employed in modern and cutting-edge information technology (IT) systems. When sustainability principles are incorporated into requirements engineering, software design priorities can change and address current design issues such as energy and resource consumption, modularity, maintainability, and adaptability. In contrast to other green approaches that consider sustainable development, there is a further need to investigate the relationship between software development and the relevant green principles of sustainability during the requirements engineering phase. We present a new mechanism for mapping software nonfunctional requirements (NFRs) to defined dimensions of green software sustainability, consisting of two mapping steps: 1) between NFRs and sustainability dimensions; and 2) between sustainability dimensions and two clusters of green IT aspects defined in this work. The overall architecture of the promising approach is based on the use of the Bidirectional Encoder Representations from Transformers (BERT) language model with an expanded dataset. We consider transfer learning and domain-specific fine-tuning capabilities for constructing and evaluating a model specifically tailored for developing a proof of concept of the greening software requirements engineering task, as language models have recently emerged as a potent technique in the field of software engineering, with numerous applications in code analysis, automated documentation, and code generation. In addition, we test the model's performance using an extended version of the PROMISE_exp dataset after adding a new binary classification column for categorizing sustainability dimensions into two defined clusters: Eco-technical and Socioeconomic, and having a selected domain expert label the raw data. The model's efficiency is evaluated using four matrices—1) accuracy; 2) precision; 3) recall; and 4) F1 score—across a variety of epoch and batch sizes. Our numerical results demonstrate the viability of the approach in text classification tasks via performing well in mapping NFRs to software sustainability dimensions. This acts as a proof of concept for automating the sustainability measurement of software awareness at the early development stage. In addition, the results emphasize the importance of domain-specific fine-tuning and transfer learning for obtaining high performance in classification tasks in requirements engineering.

INDEX TERMS Green software engineering, requirements engineering, sustainable software system, green IT, language model, BERT, non-functional requirements classification.

I. INTRODUCTION

Given the rapid development of technologies and the need to manage massive amounts of data, perform multiple tasks simultaneously, and provide high levels of performance and reliability, adopting modern information technology (IT) systems with the appropriate information and communication technology (ICT) infrastructure has become a necessity. Furthermore, IT's substantial role in several domains, such as virtual meetings, smart transport systems, decision-support

The associate editor coordinating the review of this manuscript and approving it for publication was Hailong Sun.

systems, intelligent health care applications, robotics, and smart Internet of Things systems used in smart cities, highlights its importance to many fields. Thus, some software is utilized in organizations for ICT realization, in which these software play important role in service-providing and production processes. Notably, companies that manufacture hardware and develop software have focused on the design and implementation of these and have largely ignored the sustainability of the resources used to create them. This lack of attention has resulted in a significant problem that can be analyzed from two perspectives: 1) IT system (software); and 2) sustainability [1]. On the IT systems side, the consequence is the exponential growth of software complexity, which has negative effects on the owner, stakeholders, and costs. Conversely, IT systems have negative effects on sustainability, including in terms of the quantity of hardware; data centers; and the energy required to operate, maintain, and develop these systems' processes. In addition, the negative effects on the economic and social dimensions cannot be overlooked [1], [2].

This increases the demand for ''green and sustainable'' software—that is, software that has a very low impact on environment, social, and economic aspects, such as performance level, network bandwidth, and hardware requirements that directly lead to consuming more energy or natural resources. To address this demand, greening software engineering research can be a core solution to enhance the software system sustainability from different perspectives. Sustainable software development is a key point of view that must be considered to achieve software with minimal impacts on the economy, society, and the environment.

The work presented here contributes to the field of software engineering and sustainability in the following ways. First, by proposing a proof-of-concept approach to greening requirements engineering, a crucial stage of the software system development life cycle (SDLC). Although several studies have been conducted on software requirements classifications using various natural language processing (NLP) and machine learning (ML) techniques, such as [3], [4], and [5], our study extends their work by mapping software nonfunctional requirements (NFRs) to dimensions of green software sustainability via a defined mechanism.

Second, our work expands the widely used PRedictOr Models in Software Engineering (PROMESE) dataset with a new feature representing the author-defined dimensions of software sustainability. This dataset is utilized for training the language model developed in this work. To the best of our knowledge, this is the first study to demonstrate such a correlation between NFRs and sustainability dimensions at dataset level.

Third, we develop, use, and evaluate an efficient fine-tuned Bidirectional Encoder Representations from Transformers (BERT) language model for classifying different types of NFRs into appropriate sustainability dimensions. This model is considered a core component of our promising approach to measure the degree of sustainability awareness and consideration at the requirements engineering stage of the SDLC.

Notably, various pretrained language models are utilized in different NLP tasks, such as the Generative Pre-trained Transformer (GPT) developed by OpenAI, which uses an unsupervised learning approach for text generation [6]; the Text-to-Text Transfer Transformer (T5) developed by Google, which is trained on text translation, summarization, and other text-to-text NLP tasks using a unified framework [7]; and the Robustly Optimized BERT Pretraining Approach (RoBERTa) introduced by Facebook as a BERT modification, which uses additional training data and techniques to improve the overall performance [8].

We use the BERT model in this study, based on the developed mapping task from NFRs into sustainability dimensions of software and the features of the data in the selected dataset. BERT is considered a highly effective pretrained language model that has been revealed to perform well on a broad range of NLP tasks, especially in fine-grained classification tasks aimed at classifying text into categories [6]. Unlike other language models, the accuracy of the fine-grained classification task gains more benefit from BERT's bidirectional nature and attention mechanism. In addition, because BERT was pretrained on large amounts of data from a variety of contexts, a strong foundation is leveraged for transfer learning to new tasks in different contexts. This makes BERT suitable for fine-tuning a small dataset such as that selected in this study [6].

The remainder of this paper is organized as follows. Section II presents the research background and a literature review. Section III describes the training dataset, the dataset expansion and preprocessing stages, and the feature engineering process. Section IV presents and discusses the major characteristics of the two-step mapping approach proposed in this study. This approach involves categorizing: 1) software NFRs into sustainability dimensions for software systems; and 2) sustainability dimensions into green IT factors. Section V demonstrates the experiment conducted to evaluate the proposed for BERT model and discusses the results. Lastly, Section VI discusses the conclusions of this study and future research directions.

II. GUIDELINES FOR MANUSCRIPT PREPARATION

This section represents a brief discussion on the foundations of this work, including state-of-the-art sustainability in software engineering and the ways in which ML is utilized for classification in requirements engineering.

A. SUSTAINABILITY IN SOFTWARE ENGINEERING

IT sustainability typically involves hardware sustainability, and software sustainability often takes the form of ''greening'' IT. ''Greening IT'' [9] refers to the research and practice of designing, manufacturing, utilizing, and disposing of computers, servers, and associated subsystems efficiently while ensuring a negligible or zero impact on the environment. Introspection is encouraged throughout the green software development process to lessen the software's

ecological footprint. However, software sustainability is rarely discussed in the software engineering domain from the perspective of software requirements.

Software systems are the core of this new and developing technologically organized world, and their uses vary widely—from writing a document on a computer to controlling space rockets. Software must be sustainable because people rely on it for daily activities. Sustainability has been designated a primary challenge in scientific and engineering software development as the world moves toward new paradigms in research and computing infrastructures.

The Software Sustainability Institute defines ''sustainability'' as ''the availability of software that will continue to be enhanced and supported into the foreseeable future'' [10]. Although the definition is ambiguous, it suggests that sustainability relates to software availability, extensibility, and maintainability, and these attributes can be aligned with the definitions provided by the Institute of Electrical and Electronics Engineers [11]. In this regard, software development is a complex task performed in an environment of perpetual change and uncertainty, resulting in unsustainable software products. Thus, designing sustainable software—that can provide the required output and perform the required tasks more efficiently for a longer duration than unsustainable software—will reduce the time, computational resources, money, and labor invested in development.

1) WHY SUSTAINABLE SOFTWARE DEVELOPMENT?

In 2015, the United Nations (UN) [12] outlined 17 sustainable development goals (SDGs) and 169 targets as a part of the UN's 2030 Agenda for Sustainable Development. These cover three sustainability dimensions: 1) environmental; 2) social; and 3) economic. Correlations between the SDGs and ICT are discussed in [13]. For the three sustainability dimensions, several challenges were identified in achieving these goals in the three dimensions. Consequently, sustainability becomes essential to include in software development, and there is a demand for more attention on raising awareness about, and how to innovate, green and sustainable software to support achievement of the SDGs by 2030. Thus, the identification of software development sustainability requirements is considered a crucial topic that has been investigated and discussed in various countries, such as Brazil [14], [15], Pakistan [16], and Saudi Arabia [17], [18].

According to [19], [20], there is no common understanding about the definition of sustainability in the context of requirements engineering. It refers to the systematic process of designing, implementing, and maintaining software artifacts over prolonged durations. This process is expected to exhibit efficiency, responsibility, and a flexible nature that accommodates technological advancements and evolving user needs. Concurrently, it should aim to mitigate adverse repercussions on the environment, economy, and societal structures.

The three sustainability dimensions, and more (technical and individual), have been discussed in relation to greening the software development process, including the requirements engineering and design and implementation phases [21]. This exploratory study found that most software professionals agreed that the sustainability of software is important and beneficial, and that all sustainability dimensions, not just the environmental effects, must be considered when the impact of software systems is investigated [21].

2) DIMENSIONS OF SOFTWARE SUSTAINABILITY

The dimensions of software sustainability are based on technical and environmental aspects. Long-lasting software affects the economy because such software focuses on preserving capital and financial value. It also ensures the long-term use of software systems and their appropriate evolution in a constantly changing execution environment.

Software-intensive systems encompass the direct support of social communities in any domain, as well as processes that generate indirect long-term benefits for the communities. Environmentally, sustainability aims to enhance human welfare while protecting natural resources. This approach entails addressing ecological requirements, such as energy efficiency and ecological awareness, as the resource base expands.

- Environmental Sustainability

''Environmental software sustainability'' involves developing and maintaining software systems that minimize ecological effects, promote resource efficiency, and ensure long-term software viability. For instance, energy-efficient algorithms and data storage methods reduce energy consumption [22].

- Social Sustainability

''Social software sustainability'' is the design, development, and maintenance of software systems that promote positive social effects, inclusivity and accessibility, and long-term benefits to users and communities, such as a collaborative platform that promotes equal access to information and learning resources for people of all socioeconomic backgrounds [23].

- Economic Sustainability

''Economic software sustainability'' involves the development, deployment, and maintenance of software systems that promote financial viability, long-term growth, and cost-effective solutions for stakeholders. For instance, open-source software reduces licensing costs and fosters innovation through collaborative development [21].

- Technical Sustainability

''Technical software sustainability'' ensures long-term functionality, adaptability, and reliability by using modular architecture and best practices for maintainability and extensibility in software projects [21].

3) GREEN INFORMATION TECHNOLOGY

''Green IT'' is the practice of designing, producing, utilizing, and disposing of computers, servers, peripherals, networking devices, and other IT-related hardware in an environmentally responsible and sustainable manner [24]. Green IT seeks to mitigate the negative environmental effects of technology, reduce resource consumption, and encourage the efficient and responsible use of technology within organizations and

TABLE 1. Green information technology aspects and related software sustainability concerns.

Green IT Aspect	Software Sustainability Concerns
Energy Efficient	Improving the energy efficiency of both IT equipment operation and computational operations. This includes instituting energy-saving practices in data centres, such as efficient cooling systems and server virtualization, and utilizing energy-efficient hardware and software.
Resource utilization	maximizing the use of devices and software throughout the production, operation, and disposal of IT equipment. This includes the development of software and systems that are resilient, scalable, and readily upgradable and expandable.
E-waste management and recycling	Minimizing the environmental impact of electronic waste while promoting resource conservation. This involves discarded electronic devices or their components, such as old computers, screens, Storage devices, batteries, printers and more.
Sustainable procurement	Adopting sustainable procurement practices, such as purchasing economically viable and user-friendly IT products; selecting suppliers with strong sustainability credentials; and taking into account the longevity and user productivity of IT equipment when making purchasing decisions.
Energy Efficient	Involving all organizational initiatives that seek to equip employees with the knowledge and skills necessary to adopt socially and economically sustainable practices. This enables organizations to implement changes, reduce expenses, and increase employee productivity.
Resource utilization	Establishing guidelines to enable businesses to enhance their reputation, minimize costs and risks, raise revenues and contribute to the well-being of their employees and the society.
E-waste management and recycling	Improving the energy efficiency of both IT equipment operation and computational operations. This includes instituting energy-saving practices in data centres, such as efficient cooling systems and server virtualization, and utilizing energy-efficient hardware and software.
Sustainable procurement	maximizing the use of devices and software throughout the production, operation, and disposal of IT equipment. This includes the development of software and systems that are resilient, scalable, and readily upgradable and expandable.
Energy Efficient	Minimizing the environmental impact of electronic waste while promoting resource conservation. This involves discarded electronic devices or their components, such as old computers, screens, Storage devices, batteries, printers and more.
Resource utilization	Adopting sustainable procurement practices, such as purchasing economically viable and user-friendly IT products; selecting suppliers with strong sustainability credentials; and taking into account the longevity and user productivity of IT equipment when making purchasing decisions.
E-waste management and recycling	Involving all organizational initiatives that seek to equip employees with the knowledge and skills necessary to adopt socially and economically sustainable practices. This enables organizations to implement changes, reduce expenses, and increase employee productivity.
Sustainable procurement	Establishing guidelines to enable businesses to enhance their reputation, minimize costs and risks, raise revenues and contribute to the well-being of their employees and the society.
Energy Efficient	Improving the energy efficiency of both IT equipment operation and computational operations. This includes instituting energy-saving practices in data centres, such as efficient cooling systems and server virtualization, and utilizing energy-efficient hardware and software.
Resource utilization	maximizing the use of devices and software throughout the production, operation, and disposal of IT equipment. This includes the development of software and systems that are resilient, scalable, and readily upgradable and expandable.
E-waste management and recycling	Minimizing the environmental impact of electronic waste while promoting resource conservation. This involves discarded electronic devices or their components, such as old computers, screens, Storage devices, batteries, printers and more.

society. Table 1 presents six green IT aspects and indicates green IT's relevance to software sustainability concerns as obtained from the background reading.

Software sustainability may boost green IT by encouraging the development of creative, efficient, and environmentally friendly solutions that contribute to a more sustainable future. It affects green IT practices from different perspectives [21]. For example, environmentally sustainable software requires less energy and resources and provides a smaller IT system carbon footprint; economically sustainable software optimizes costs and resource allocation and extends the hardware life cycle, yielding long-term benefits for enterprises; and socially sustainable software encourages developers, end-users, and other stakeholders to embrace responsible software practices that support sustainability goals [21], [22].

B. NONFUNCTIONAL REQUIREMENTS

''Nonfunctional requirements'' are essential constraints that define the overall characteristics and behavior of a system, ultimately determining the system's quality, performance, and user experience. They are not related directly to the software functionalities but, rather, describe how well the system performs its functions [23]. They have been characterized in various ways depending on how they are contextualized, instantiated, or met. NFRs serve as reasons for design decisions and limit how the needed functionality can be realized [24].

NFRs are classified into several types, each addressing a different aspect of software performance, and specify a broad range of system qualities, including: performance requirements, which dictate response times, throughput, and resource usage; reliability, which concerns fault tolerance and software availability; usability, which focuses on user-friendliness, learnability, and accessibility; maintainability, which includes modifiability, testability, and extensibility; and security, which targets cohesion [24], [25], [26]. Table 2 summarizes the common NFR types discussed in this subsection.

The term ''sustainability'' has been infrequently used in the software development domain until recently' [27]. Introduced as a method to analyze software sustainability, the Sustainability Analysis Framework aims to capture the relevant qualities that characterize the sustainability related to software systems, assisting in identifying how these qualities influence each other regarding the various aspects of sustainability. ''Nonfunctional properties'' have been defined as software qualities. To facilitate holistic reasoning and decision-making regarding software, hardware, and human

TABLE 2. Types of software nonfunctional requirements and their definitions [26], [27], [28], [29].

NFR	Definition
Reliability (RE)	The ability of a software to perform its intended functions without failures or errors in a specific time, under some determined conditions.
Usability (US)	To what extend the software is user-friendly, ease to adopt and easy to use that allows achieving user goals with efficiency, satisfaction and effectiveness.
Efficiency (EF)	The amount of computing and program processing required to execute a function.
Portability (PO)	The ability of the software to run under various computing environments.
Interoperability (IN)	The ability of the final system to couple all its components together without problems, by using standard data representation format and highly independent and compatible modules.
Maintainability (MN)	To what extend the software is traceable and requires short time for fixing errors, increasing performance, adapting or changing environment, improving other operational qualities and functionality, or adapted to changing requirements.
Scalability (SC)	The property of software to increase its ability and functionalities based on the user needs, workloads or data volumes without effects on performance.
Accessibility (AC)	The property of software to be usable by specified users to achieve their goals with effectiveness, efficiency, convenience fulfilment in a context.
Security (SE)	The property of protecting the system and its data against attacks, including unauthorized access, disclosure and destruction; To ensure the integrity, confidentiality, and availability of the software.
Functional Suitability (F)	The software system's capabilities, functions, and features that ensure fulfilling its intended purpose and meets the needs of its users.
Availability (A)	To what extend the final software is available and ready to do its tasks for users at the time of use.
Fault Tolerance (FT)	The property of software to detect and recover from faults and operated as required in spite of the presence of errors, failures or unexpected conditions associated to its hardware or system.
Look and Feel (LF)	The property of software to be perceivable and attractive by users and comply with corporate branding standards.
Operational (O)	The property of software system performs effectively and efficiently in its intended environment, from different stakeholder perspectives.
Performance (PE)	The property of software system to be work effectively with respect to several factors including time constraints, processing time, throughput and allocation of resources.
Legal (L)	The property of software adheres to applicable laws, industry standards, and organizational policies, minimizing the risk of legal disputes, penalties, or reputational damage.

and system factors, various approaches to assessing the quality of software architecture, in particular, have been developed.

NFRs are also utilized to validate and verify the completion of the software system and its success in fulfilling the user needs [31]. Thus, automatic analysis of NFRs can have great influence on the software development lifecycle. When NFRs are poorly defined, misunderstood, or neglected during the development process, the resulting software product can suffer in several ways, such as security vulnerabilities, lack of scalability, poor usability, maintenance difficulties, and various performance and reliability issues [32], [33].

C. SOFTWARE QUALITY MODELS

Software quality can be divided into two categories: 1) process; and 2) product [34]. The technology, people, tools, and the organization all contribute to software process quality, whereas software product quality comprises documentation clarity and integrity, design traceability, application reliability, and test integrity. Historically, in 1977, the United States Air Force adopted McCall's quality model [35]. Quality attributes (quality factors, criteria, and metrics) are defined by the hierarchical factor criteria metric model [36]. To connect consumers and software developers, the model focuses on user- and developer-centric factors [37], [38]. McCall's model defined software quality as product operation, revision, and transfer and included 11 quality parameters [36], [38]. The criterion and quality components in McCall's quality model are tree style [26]. In 1978, Boehm introduced another software quality model, which is commonly used [38]. This quality model is considered superior to McCall's model, despite the models' structural similarities. This quality model identified 23 nonfunctional software quality factors [35]. It prioritized users, developers, testers, and maintainers, and addressed problems in existing quality models, which quantified software quality [38].

Software is defined by qualities and metrics, which has high, midrange, and primitive characteristics [38]. The ISO/IEC JTC 1/SC 7 Software and Systems Engineering standard is discussed in [39]. This standard provides software rules and measurements to develop a model [36] based on the McCall, Boehm, and functional and NFR quality models. ISO/IEC TR 9126-4:2004 defines quality in use, internal quality, and process quality. The model has six product quality characteristics (portability [PO], maintainability [MN], efficiency [EF], functional [F], reliability [RE], and usability [US]), which have been further separated [38], [39], and 21 software quality criteria [39], [40], [41].

D. MACHINE LEARNING FOR REQUIREMENTS ENGINEERING

ML research creates computer programs to learn from data. Several software engineering studies have employed ML methods to identify software engineering stages in the SDLC. In addition, requirements specifications studies have used different ML techniques to automate requirements classification and reduce its challenges [10], [39], [42], [43], [44].

In [43], a novel classification model for software requirements specifications (SRS) based on the BERT model is presented. The transfer learning capabilities of BERT has

been used to train a classifier on SRS documents, resulting in superior performance compared with conventional ML techniques. They demonstrate that the BERT-based model can accurately categorize SRS text into various categories, including functional, nonfunctional, and quality requirements. In [44], the nonfunctional and functional Requirements classification using BERT (NoRBERT) language model's transfer learning-based capabilities are utilized to investigate the challenge of classifying functional and nonfunctional SRS documents. The work concluded that NoRBERT also significantly enhances prediction performance compared with other deep-learning techniques.

Another keyword classification technique as an NFR classifier is discussed in [45]. The classifier solved the NFR classification problem by discovering a set of weighted indicator terms for each NFR type using a training set. This strategy restricts the NFR classifier to only recognizing and retrieving categories for which it has been trained.

This classifier has two phases. First, indicator terms for each NFR category are selected. This phase requires the use of pre-classified training requirements. The training set requirements calculate a probabilistic weight for each potential indicator phrase for each NFR type. An indicator term's weight represents its requirement type. For example, terms such as ''authenticate'' and ''access'' that appear frequently in security requirements and rarely in other types of requirements are strong indicators for security NFRs, whereas terms such as ''ensure'' that appear less frequently in security requirements or appear in various requirement types are weaker indicators.

In the second phase, indicator terms can categorize additional requirements and declarations. Each requirement's indicator terms are used to calculate a probability value for the new requirement's NFR type. Multiple indicator terms of a particular NFR type classify a requirement. Unclassified needs are functional unless they score above a threshold for a particular NFR type. The authors in [43] set a threshold to achieve high recall, since classification findings are only successful if a significant proportion of target NFRs are discovered for a certain type.

Furthermore, a sentence-based automatic requirements document classifier has been described [46]. The classifier was implemented using the General Architecture for Text Engineering system [47]. A custom text-mining infrastructure was used to identify and classify candidate sentences using ML algorithms. The presented classifier was trained on both the PROMISE and the Concordia Requirements Engineering (RE) corpus. In the experiments, presented in [47], third-order polynomial support vector machines fared best. Sentence token stem-based unigrams were used for training. ''Web-based displays of the most recent ASPERA-3 data shall be made available to the public'' was annotated as both a functional requirement (FR) and a design constraint.

In conclusion, unlike what is proposed in our work, the discussed studies [43], [44], [45], [46], [47] focused on NFR classification without considering the correlation between NFR types and dimensions of software sustainability.

E. ENGLISH LANGUAGE PROCESSING IN PYTHON

The field of NLP has seen the development of several powerful Python libraries, each designed to cater to different aspects of text processing. Among these, the Natural Language Toolkit (NLTK) and spaCy are two of the most prominent and widely used tools for text processing [48], [49]. NLTK is a comprehensive library that provides extensive resources, including corpora, lexical databases, and pretrained models, as well as a variety of text-processing functions, such as tokenization, stemming, lemmatization, part-of-speech tagging, and parsing. Developed as an open-source project, NLTK has been widely adopted in academic and educational settings, primarily because of its ease of use and extensive documentation. As a result, it has become an invaluable resource for those learning and teaching NLP [45].

SpaCy [49] is a more recent NLP library that concentrates on advanced and efficient text processing, making it ideal for large-scale information extraction projects. Users can promptly develop and deploy NLP applications with the help of spaCy's simple application programming interface (API), quick performance, and pretrained models for multiple languages. Built with production-ready applications in mind, spaCy prioritizes efficiency and efficacy over the NLTK library's comprehensiveness. As a result, spaCy has acquired traction in the industry, becoming a popular option for advanced and practical NLP applications [50]. While both NLTK and spaCy provide potent tools for text processing, the choice between the two libraries largely depends on the task's specific requirements, complexity, and scope.

III. TRAINING DATASET AND PREPROCESSING

We considered the expanded version of the commonly used PROMISE corpus [54] (PROMISE_exp), presented in [50], as a base for this research. We added different types of software requirements after searching in requirements engineering documents, available online, using the Google search engine. In addition, experts applied a validation process to measure the extracted requirements, before adding new requirements to the repository. This expansion improved the original dataset as was revealed on evaluating the new version against the original repository using the ML techniques of a support vector machine, decision tree learning, the k-nearest neighbors algorithm, and the multinomial naive Bayes algorithm.

In contrast to the original PROMISE repository that has only 625 requirements, PROMISE_exp consists of 969 labeled software requirements expressed using natural language (English). It has been used in several studies, such as [52], [53], [54], [55], and [56], to address the problem of software requirements classification. Both versions of the dataset consist of three main attributes: ProjectID, RequirementText, and Class. The ProjectID attribute represents the identification number of the project from which the requirements were extracted, the RequirementText attribute represents the actual text used to express a software requirement in a particular project, and the Class attribute represents the categorized type for each RequirementText.

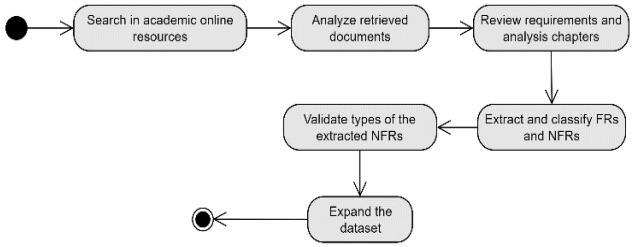


FIGURE 1. Dataset expansion and labelling strategy.

PROMISE_exp classifies software requirements into FRs, and 11 types of NFRs: usability (US), portability (PO), legal (L), maintainability (MN), scalability (SC), security (SE), availability (A), fault tolerance (FT), look and feel (LF), operational (O), and performance (PE).

A. EXPANDING PROMISE-EXP DATASET

To address all the quality attributes presented in the proposed software sustainability measuring framework, and as a part of this study's contribution, we extended the PROMISE_exp dataset in two stages. First, we considered three types of NFRs—RE, EF, and accessibility (AC)—and added these into the dataset, because these types of software NFRs can also be used to determine the degree of sustainability of a software system. In addition, to address more NFRs that are not considered in the PROMISE_exp version of the dataset, including AC, EF, interoperability (IN) and RE, we added numerous instances to the dataset. The Unified Modeling Language (UML) activity diagram shown in Fig. 1 illustrates the steps taken to create the extended dataset, including the data preprocessing steps taken to clean, transform, and label data.

As shown in Fig. 1, some steps were performed before adding a new NFR or FR instance into the dataset. In the first step, publicly available documents of student graduation projects and dissertations enrolled in a computing-related degree or program in past years were retrieved through an online search. Then, all retrieved dissertations and project reports were analyzed, following which only project and dissertation documents based on the software/system development theme were considered and the remaining documents discarded.

Next, the software/system requirements and analysis chapters of the documents considered were reviewed and requirement sentences were determined and extracted. Most of the requirements identified from the documents were classified according to whether they are functional or a type of NFR. Only a few NFR sentences were manually classified into one of the NFR types considered.

After completing this NFR extraction and classification step, all selected NFRs, as well as FRs, were validated and checked by a software engineer / computer scientist peer for linguistic ambiguity and correctness. Then, all validated software/system requirements were added to the dataset as new instances. The expanded version of the PROMISE_exp dataset utilized in this study consists of 1149 requirements

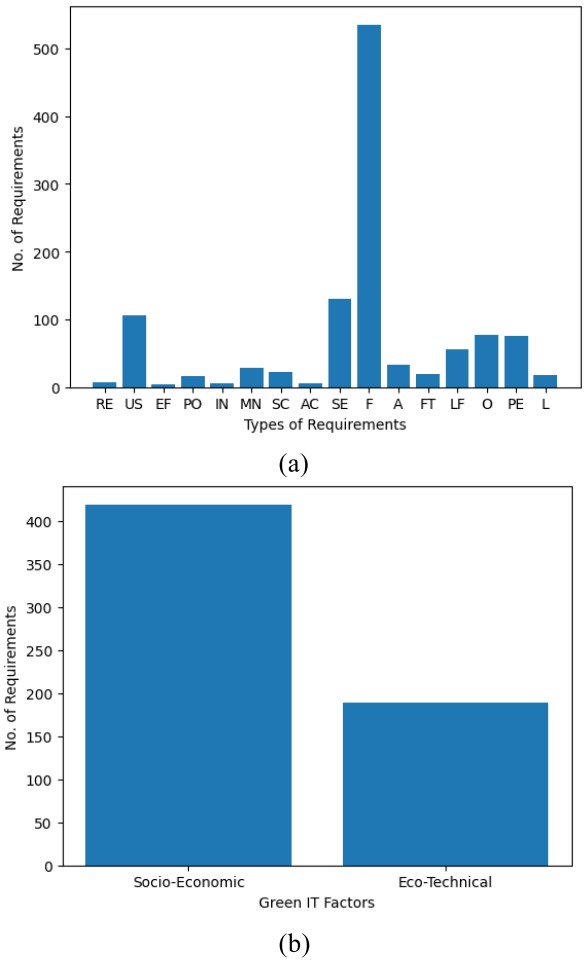


FIGURE 2. (a) Distribution of nonfunctional requirements in the expanded dataset. (b) Distribution of label values in the dataset after applying the labeling process.

distributed over 69 documents. Its composition is presented in Fig. 2(a).

As part of the dataset expansion process, the current label column in the dataset was considered another feature of the dataset. The new expanded version was considered unlabeled, thus a suitable labeling technique was applied, as discussed next.

B. DATASET LABELING TECHNIQUE

Based on the nature of the expanded dataset version utilized, the manual labeling technique was considered suitable for annotating it. A domain expert was selected to go through each line of the raw data and assign an appropriate label. The labels have been used as a meaningful and informative way to represent the context of software sustainability associated with common green IT practices. The activity diagram displayed in Fig. 3 demonstrates the overall labeling process.

1) LABEL IDENTIFICATION

In line with the strategy adopted to map sustainability dimensions onto green IT aspects (discussed in Section IV-B), two informative labels have been introduced for representing

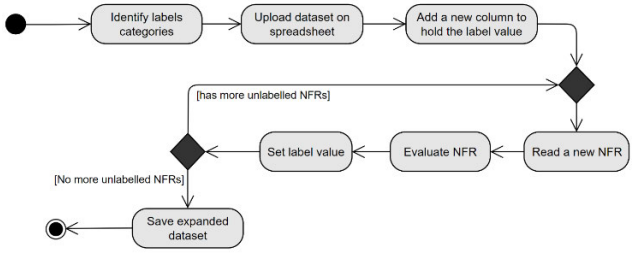


FIGURE 3. Labeling process in the utilized dataset.

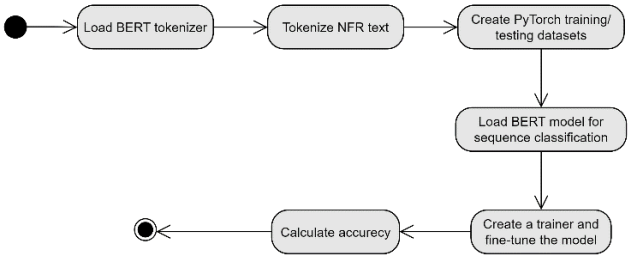


FIGURE 4. Bidirectional Encoder Representations from Transformers (BERT) model development and training process.

two categories of green IT factors: Eco-technical and Socioeconomic. These new labels were used for applying an appropriate binary semantic text classification technique, which classifies semantically relevant NFRs into one of these categories.

The identification process started by asking the selected expert to make judgments about a given unlabeled NFR statement. The labelers was asked to label all the NFR statements in a dataset where ''does the statement indicate or imply a direct contribution to technical or environmental sustainability'' is true. For instance, the statement ''The application responds in a reasonable time limit'' contributes to technical and environmental sustainability by not consuming a large amount of power because of slow performance. Subsection B, ''Feature Engineering for Green Information Technology Aspects,'' discusses the direct influences of each type of NFR on different sustainability dimensions, which were used as guidance for the labeling process. After this task, the labels were distributed in the expanded dataset as shown in Fig. 2(b).

2) NONFUNCTIONAL REQUIREMENT EVALUATION

The evaluation process consisted of three manual steps, performed by a selected domain expert: 1) evaluating the direct influence of each NFR appearing in the expanded dataset on a sustainability dimension; 2) evaluating the semantic meaning of each NFR and its ability to influence green IT; and 3) filling the newly added column with a suitable label value.

C. FEATURE ENGINEERING FOR GREEN INFORMATION TECHNOLOGY ASPECTS

This section proposes the systematic process of features engineering that aims to obtain linguistic features of textual sustainability requirements based on many NFRs and quality attributes considered in the extended dataset. From among the various feature engineering techniques available, the BERT technique, provided by Google AI, was adopted. Building the BERT model can facilitate a more precise comprehension of the text's word semantics and syntactic relations [5]. BERT is pretrained utilizing two unsupervised learning goals, masked language modelling and next sentence prediction. These objectives allow BERT to acquire a profound comprehension of the language structure and semantics without relying on labeled data, making it highly adaptable for the binary text classification required in the proposed approach [61], [62].

With its transfer learning capabilities, BERT can replace traditional pipelines in NLP-based systems [5]. With minimal fine-tuning, BERT models can be applied to tasks other than the one for which they were trained. Thus, developing BERT may require less effort than performing traditional NLP processing using a deep-learning technique, because loading a pretrained BERT model and performing fine-tuning tasks are all that are required. As a result of its transformer-based architecture, the BERT paradigm provides more efficient computation and parallelism when dealing with long-range text dependencies. These advantages surpass those of other techniques, including bag of words, term frequency–inverse document frequency, word embedding, recurrent neural networks, and long short-term memory [5].

In this study, the Python library scikit-learn (sklearn) was used to implement the data preprocessing and feature extraction strategy. PyTorch was also used—another powerful versatile library for tensor computation in Python, PyTorch offers a dynamic and user-friendly approach for model deployment and dataset training.

The UML activity diagram shown in Fig. 4 demonstrates the NLP pipeline of BERT designed in this study. First, the dataset is converted into the PyTorch data frame, and split up into training and validation sets. Then, the BERT tokenizer is uploaded and used to convert each NFR text into BERT tokens. All labels are transferred into numeric values. After this, training and testing datasets are created. Next, the sequence classification step is applied using the loaded BERT model. Then, the trainer is created after determining appropriate training arguments to fine-tune the BERT model. Lastly, the overall BERT classifier accuracy is calculated to be used for further predictions.

IV. PROPOSED APPROACH

Prior to an in-depth discussion of the mapping stages of the suggested approach, a concise overview of the methodology utilized in this work is necessary. The method is novel in that it addresses the problem of linkage between NFRs and the defined green IT aspects by employing well-known sustainability dimensions and the subsequent steps: 1) identifying the mapping between the four sustainability dimensions and NFRs; 2) identifying the mapping between the four sustainability dimensions and green IT aspects, which are the defined (clustered) sustainability dimensions presented in this work; 3) developing and training the BERT

model; and (4) expanding and labeling the dataset with an additional green IT characteristic.

Moreover, to the best of our knowledge, there is no dataset containing sustainability features correlated with NFRs, nor is there a comparable approach with which to compare our method. This is a further advantage of the proposed method.

A. MAPPING NFRs TO SUSTAINABILITY DIMENSIONS

As described earlier, software NFRs have a significant relationship with overall green sustainability, because they can influence the environmental impact of a software system directly and indirectly. By considering the direct influence of NFRs on software sustainability via its variant dimensions, more sustainable and environmentally friendly software can be developed. The following sections discuss the mapping of each software sustainability dimension to each NFR type that has only direct influence on the overall sustainability. We are aware of the other types of NFRs that have less influence, or indirect effects, on sustainability, but do not include these in the tables presented in this paper.

1) MAPPING ENVIRONMENTAL SOFTWARE SUSTAINABILITY TO RELEVANT NONFUNCTIONAL REQUIREMENTS

The environmental sustainability of software systems is highly dependent on NFRs, or the so-called software quality attributes, such as performance, security, and scalability. As the use of environmental software applications for monitoring, predicting, and managing environmental processes increases, ensuring that these systems are resource and energy efficient becomes essential. Moreover, performance optimization can also lead to reduced computational requirements and energy consumption, contributing to a smaller environmental footprint for these applications. In this study, we develop a detailed mapping between NFR types and environmental software sustainability after evaluating the definition of the environmental sustainability of software against the definition of each NFR type. Only NFR types that have direct influence on environmental sustainability are considered (see summaries in Table 3).

2) MAPPING TECHNICAL SOFTWARE SUSTAINABILITY TO RELEVANT NONFUNCTIONAL REQUIREMENTS

NFRs are essential to the technical viability of software. As software systems and applications become increasingly complex and embedded in multiple domains, ensuring their long-term technical viability is crucial. This involves enabling systems to adapt to changing requirements and evolve over time, while mitigating the risk of technical obsolescence. Focusing on the capacity of software systems to adjust to new technologies, platforms, and environments is considered vital to maintaining technical relevance in a rapidly changing technological landscape. In addition, promoting the reuse of existing resources ultimately contributes to the technical sustainability of software and systems [59], [60]. Table 4 shows the detailed mapping between NFR types and the technical sustainability of software after evaluating the definition of technical sustainability of software against the definition of each NFR type. Only NFR types that have direct influence on technical sustainability are considered in the proposed approach.

TABLE 3. Types of nonfunctional requirements and their direct influence on the environmental sustainability of software.

NFR	Direct Influence
PE	• Reducing energy consumption when performing tasks. • Reducing the amount of time required to complete tasks. • Minimizing the CPU, memory, and network bandwidth usage.
EF	• Reducing energy consumption with efficient processes.
IN	• Minimizing the need for redundant hardware and software, which lead to less energy consumption.
FT	• Reducing the need to perform a complete system restarts or extensive recovery processes for handling failures or errors. • Ensuring efficient use of resources during failures or disruptions. • promoting the use of environmentally friendly infrastructure.
MN	• Reducing the need for frequent software replacements, which is leading to fewer discarded software products and a reduced demand for new hardware. • Declining in the disposal of electronic devices and components.

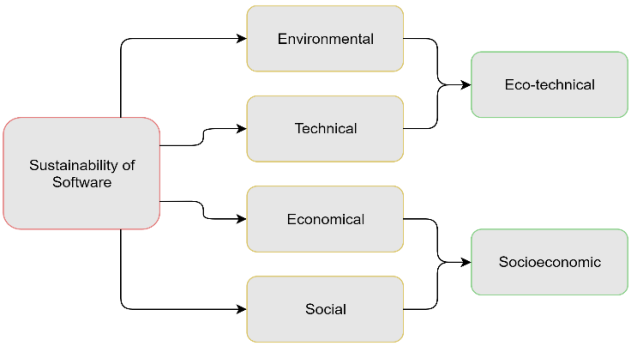


FIGURE 5. The defined groups of the common four sustainability of software dimensions used for training the developed BERT model.

B. MAPPING SOFTWARE SUSTAINABILITY DIMENSIONS INTO GREEN IT ASPECTS

As mentioned earlier, the various dimensions of sustainable software have a significant impact on the general green IT practices from different viewpoints. In the proposed mapping, the four dimensions of software sustainability are clustered into two groups (illustrated in Fig. 5): 1) the *Eco-technical* factor combines the technical and environmental (ecological) dimensions of sustainability—this factor emphasizes the effective efficient development and application of technology in a way that promotes the environmental sustainability and efficiency of software; and 2) the Socioeconomic factor, which combines the social and economic dimensions of sustainability, and refers to the interrelationship between social and economic factors at the individual and community levels, emphasizing the connection between the two in various contexts and considerations, as well as its effects on user behavior and organizations.

TABLE 4. Types of nonfunctional requirements and their direct influence on the technical sustainability of software.

NFR	Direct Influence
US	• Ensuring that the system can be accessed and used by a wide range of users, without being discriminated, including those with limited technology skills or resources. • Improving user satisfaction and customer loyalty over time.
AC	• Ensuring that the system can be accessed and used by a wide range of users, including those with disabilities.
F	• Enabling the utilization of technology effectively and efficiently. • Avoiding lost productivity and consuming resources unwisely.
SE	• Enabling the use of technology safely, securely, and with confidence over time.
PO	• Reducing social inequalities caused by limiting access to technology and data. • Ensuring access to technology and information for a diverse range of users. • Ensuring that more people can access and benefit from services, regardless of their technological environment.
A	• Promoting user trust, satisfaction, loyalty, and long-term engagement. • Ensuring that users can access and use services regardless of location, time zone, or connectivity constraints. • Ensuring continuity of service even in adverse conditions can minimize the negative impacts of software outages and disruptions on users and communities. • Increasing productivity by allowing accessing software services whenever needed.
LF	• Promoting user engagement and long-term adoption by making software easier to interact with and use. • Promoting user loyalty/ trust using professional, visually appealing look and feel. • Offering a positive and enjoyable user experience via establishing an emotional connection with the software. • Making the software tailored to different cultural contexts by considering language, visual elements, and cultural preferences. • Ensuring the software is relevant and appealing to users from diverse backgrounds and regions by considering localization and sensitivity.
O	• Operating software in a way that enhances user satisfaction and productivity in the workplace
L	• fostering trust and confidence among different stakeholders, leading to increased long-term adoption.

V. MODEL EVALUATION AND RESULTS DISCUSSION

In this section, we discuss the process we undertook to evaluate the performance of the BERT model, developed in this study for binary text classification. We followed a strategy of fine-tuning the developed BERT model on the dataset, which consisted of examples of NFR phrases classified into two green IT factor classes: Socioeconomic and Eco-technical. The dataset we used in our experiment was imbalanced. It comprised 608 instances of NFR textual phrases, of which 419 were labeled Socioeconomic and 189 *Eco-technical*.

A. EXPERIMENT OVERVIEW

To evaluate the proposed proof of concept, our experiment was designed to investigate the feasibility and measure the ability of our BERT model developed for binary classification in classifying the defined two classes of software green sustainability. This can be achieved by reaching the highest possible scores in different performance metrics. Notably, we were aiming to validate our proof-of-concept approach rather than to find the best language model.

For the evaluation experiment, the dataset was divided into training (70%) and testing (30%) datasets. A hyper-parameter tuning strategy was considered to optimize the performance of the developed BERT model using two critical parameters—number of epochs and batch size. The model was trained over multiple epoch numbers, ranging from 1 to 5, 8 and 12, two batch sizes (8 and 16), and Adam optimization in the training process. "Batch size" refers to the number of training samples used in a single update in the training process, which influences the model performance.

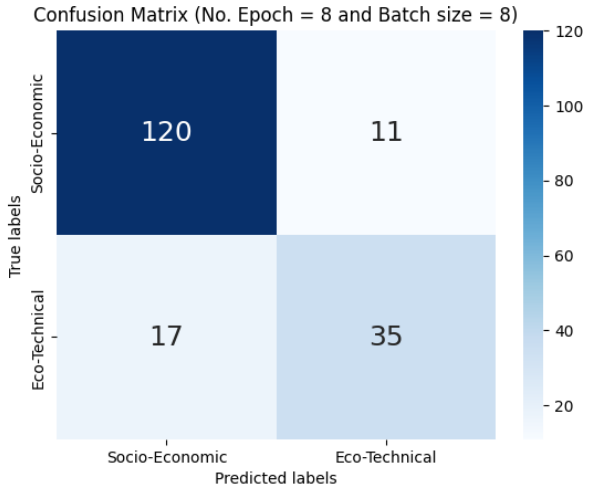
To quantify the model performance and its ability to provide correct predictions from different aspects, four common metrics—accuracy, precision, recall, and F1 score on the validation set—were considered in monitoring and examining the performance of the developed BERT model. To avoid the overfitting problem, we controlled the randomness split of training and testing data using a specific random_state value to ensure that the split is reproducible.

B. ANALYSIS AND INTERPRETATION OF RESULTS

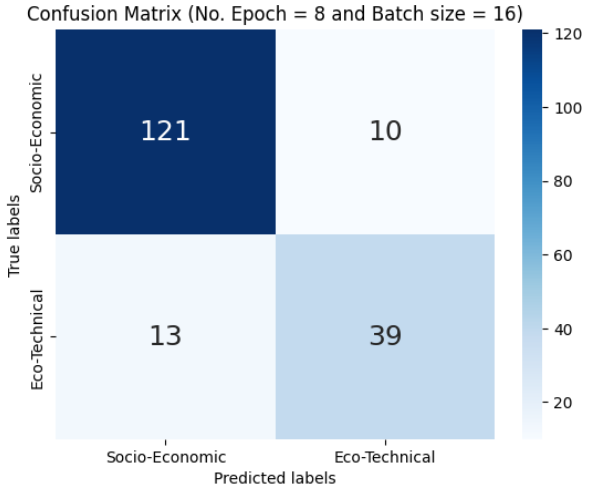
A confusion matrix was utilized to describe the performance of the classification model developed in this work. As shown in Fig. 6, we chose the Socioeconomic label as the negative and the Eco-technical label as the positive. With 8 epochs and a batch size of 8, the matrix shows that 120 instances of Socioeconomic were accurately detected, while 11 cases were predicted as Eco-technical but were Socioeconomic. Furthermore, 35 instances of Eco-technical were correctly identified, while 19 instances were predicted as Socioeconomic but were Eco-technical [Fig. 6(a)]. However, with 8 epochs and a batch size of 16, the matrix shows that 121 instances of Socioeconomic were correctly identified as Socioeconomic, whereas 10 Socioeconomic instances were misidentified as Eco-technical. Moreover, 13 Eco-technical instances were wrongly identified as Socioeconomic, while 39 Eco-technical instances were correctly identified [Fig. 6 (b)].

Tables 5 and 6 represent the performance metrics of the developed BERT model for multiple epoch numbers and two batch sizes (8 and 16). The tables reveal that the model performance improved overall, as shown by the increase in accuracy, precision, recall, and F1 score, as the number of epochs increases (from 1 to 12), for both batch sizes. In the case of a batch size of 8, the best recorded performance metric results were achieved with 8 epochs. The accuracy increases from 72.6% to 84.70% as the number of epochs increases, reflecting the improvement in the model's ability to correctly predict both green IT factor classes accurately. This is a good accuracy rate, but accuracy can sometimes be misleading, especially if the classes are imbalanced.

Regarding the model's ability to correctly predict the Eco-technical class, which is considered the positive label with a value of 1 in the label probability, the performance improves slightly as the number of epochs increases, as revealed by the



(a)



(b)

FIGURE 6. Confusion matrices representing the performance of the BERT model with different configuration of No. of epochs and batch sizes.

TABLE 5. Performance metrics with multiple epoch numbers and a batch size of eight.

No	No. of Epoch	Accuracy	Precision	Recall	F1
1	1	0.7268	0.8023	0.7268	0.0741
2	2	0.7760	0.7959	0.7760	0.3881
3	3	0.8251	0.8217	0.8251	0.6800
4	4	0.8197	0.8208	0.8197	0.6857
5	5	0.8415	0.8391	0.8415	0.7129
6	**8**	**0.8470**	**0.8432**	**0.8470**	**0.7143**
7	12	0.8361	0.8343	0.8361	0.7059

improvement of the precision ratio from 80.23% to 84.32%. Furthermore, the model identifies the Eco-technical class correctly 84.70% of the time. This percentage reflects the ratio of sensitivity metric (recall).

As the performance scores for the accuracy metric could be misleading when using an imbalanced dataset, the F1 score can be considered a better metric and is also used

TABLE 6. Performance metrics with multiple epoch numbers and a batch size of 16.

No	No. of Epoch	Accuracy	Precision	Recall	F1
1	1	0.7158	0.5124	0.7158	0.0000
2	2	0.7158	0.5124	0.7158	0.0000
3	3	0.8306	0.8353	0.8306	0.6173
4	4	0.8197	0.8187	0.8197	0.6796
5	5	0.8361	0.8309	0.8361	0.6739
6	**8**	**0.8743**	**0.8726**	**0.8743**	**0.7723**
7	12	0.8415	0.8446	0.8415	0.7290

in evaluations as it is the harmonic mean of both precision and recall [64]. Remarkable improvement of the F1 ratio can be observed (from 7.41% to 71.43%) in Table 5, which reflects the model's ability to identify Eco-technical samples accurately with a low false positive rate [64]. This score is considered reasonably good but indicates there might be some imbalance between precision and recall.

Similarly, the best recorded performance metric results are achieved with 8 epochs for a batch size of 16. The accuracy ratio increases from 71.58% to 87.43% as the number of epochs increases, reflecting the improvement in the model's ability to categorize both green IT factors accurately. Furthermore, the precision and F1 scores increase extraordinarily, as both scores grow from about 51.24% to 87.26% and from 0% to 77.23%, respectively, indicating that the performance of the developed model is satisfactory.

Summarizing, in the second scenario, with a larger batch size, all performance metric scores are improved. Specifically, the F1 ratio increases from 71.43% to 77.23%. This suggests that the model's performance in classifying both Socioeconomic and Eco-technical instances is improved.

Regarding the confusion matrices, the second scenario has fewer misclassifications. The number of false positives (instances incorrectly predicted as Eco-technical) slightly decreases from 11 to 10, and the number of false negatives (instances incorrectly predicted as Socioeconomic) decreases from 17 to 13.

C. MAIN FINDINGS

- The augmentation of the batch size from 8 to 16 leads to enhanced model performance, as evidenced by the improvement in all evaluation metrics.
- The model's capacity to correctly classify *Socioeconomic* and *Eco-technical* instances improves in the second scenario, with a larger batch size.
- The decrease in both false positives and false negatives in the second scenario indicates that the model improved its ability to make accurate predictions and avoid incorrect ones.
- In both scenarios, the number of false negatives is higher than the number of false positives. This indicates that the model is more likely to misclassify *Eco-technical* instances as *Socioeconomic* than the opposite.
- In both scenarios, the F1 ratios are slightly lower than the other metrics, which means there could be an opportunity for improvement by optimizing the model further to achieve a higher F1 score.

VI. CONCLUSION

In this study, we presented a novel two-step proof-of-concept approach of mapping software NFRs into factors that influence green IT practices. Unlike the traditional ML classification approaches used to categorize software requirements, our investigation demonstrated that the proposed mapping approach is a significant step toward understanding the semantics of NFR phrases and interpreting the corresponding sustainability purposes. Two labels that refer to green IT factors, Socioeconomic and Eco-technical factors, were defined and added to the expanded dataset to support the binary text classification task.

In addition, a pretrained fine-tuned BERT model, with its transfer learning capabilities, was developed for binary NFR classification based on the semantics of NFR statements in the greening requirements engineering approach discussed in this study. The performance of the model was monitored, analyzed, and interpreted to evaluate its capabilities for the classification process. Our evaluation experiments yielded substantial insights and outcomes. The initial model configuration, consisting of 8 training epochs and 8 batches, yielded promising results. The confusion matrix revealed a satisfactory level of accurate predictions for both Socioeconomic and Eco-technical instances.

However, to optimize the model, we experimented with a larger group size of 16 while maintaining the same number of epochs. This modification led to an improvement across all performance metrics. In addition, the confusion matrix revealed a reduction in both false positives and false negatives, indicating an enhanced capacity for prediction.

Although these results demonstrate the potential of the refined BERT model, attaining an optimal balance between precision and recall, as highlighted by the F1 score, remains a fundamental challenge. Depending on the significance of each metric in a particular application, additional model tuning may be required.

REFERENCES

[1] E. Kern, M. Dick, S. Naumann, A. Guldner, and T. Johann, "Green software and green software engineering–definitions, measurements, and quality aspects," in *Proc. ICTS*, Zurich, Switzerland, 2013, pp. 87–91.

[2] C. Calero and M. Piattini, "From requirements engineering to green requirements engineering," in *Green in Software Engineering*. New York, NY, USA. Cham, Switzerland: Springer, 2015, pp. 157–186.

[3] E. D. Canedo and B. C. Mendes, "Software requirements classification using machine learning algorithms," *Entropy*, vol. 22, no. 9, p. 1057, Sep. 2020.

[4] G. Y. Quba, H. Al Qaisi, A. Althunibat, and S. Al Zu'bi, "Software requirements classification using machine learning algorithm's," in *Proc. Int. Conf. Inf. Technol. (ICIT)*, Amman, Jordan, Jul. 2021, pp. 685–690.

[5] T. Hey, J. Keim, A. Koziolek, and W. F. Tichy, "NoRBERT: Transfer learning for requirements classification," in *Proc. IEEE 28th Int. Requirements Eng. Conf. (RE)*, Zurich, Switzerland, Aug. 2020, pp. 169–179.

[6] R. Qasim, W. H. Bangyal, M. A. Alqarni, and A. A. Almazroi, "A fine-tuned BERT-based transfer learning approach for text classification," *J. Healthcare Eng.*, vol. 2022, pp. 1–17, Jan. 2022.

[7] K. Pipalia, R. Bhadja, and M. Shukla, "Comparative analysis of different transformer based architectures used in sentiment analysis," in *Proc. 9th Int. Conf. Syst. Modeling Advancement Res. Trends (SMART)*, Moradabad, India, Dec. 2020, pp. 411–415.

[8] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "RoBERTa: A robustly optimized BERT pretraining approach," 2019, pp. 1–13, *arXiv:1907.11692*.

[9] S. Murugesan, "Harnessing green IT: Principles and practices," *IT Prof.*, vol. 10, no. 1, pp. 24–33, 2008.

[10] B. Penzenstadler, "Towards a definition of sustainability in and for software engineering," in *Proc. 28th Annu. ACM Symp. Appl. Comput.*, Salamanca, Spain, Mar. 2013, pp. 1183–1185.

[11] M. N. Malik and H. H. Khan, "Investigating software standards: A lens of sustainability for software crowdsourcing," *IEEE Access*, vol. 6, no. 1, pp. 5139–5150, 2018.

[12] United Nation. *Department of Economic and Social Affairs, the 17 Goals*. Accessed: Jul. 2, 2023. [Online]. Available: https://sdgs.un.org/goals

[13] J. Wu, S. Guo, H. Huang, W. Liu, and Y. Xiang, "Information and communications technologies for sustainable development goals: State-of-the-art, needs and perspectives," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 3, pp. 2389–2406, 3rd Quart., 2018.

[14] L. Karita, B. C. Mourão, L. A. Martins, L. R. Soares, and I. Machado, "Software industry awareness on sustainable software engineering: A Brazilian perspective," *J. Softw. Eng. Res. Develop.*, vol. 9, no. 2, pp. 2–15, 2021.

[15] L. Karita, B. C. Mourão, and I. Machado, "Software industry awareness on green and sustainable software engineering: A state-of-the-practice survey," in *Proc. 33rd Brazilian Symp. Softw. Eng.*, Salvador, Brazil, Sep. 2019, pp. 501–510.

[16] A. Javeed, M. Y. Khan, M. Rehman, and A. Khurshid, "Tracking sustainable development goals—A case study of Pakistan," *J. Cultural Heritage Manag. Sustain. Develop.*, vol. 12, no. 4, pp. 478–496, 2021.

[17] S. Aljarallah and R. Lock, "An exploratory study of software sustainability dimensions and characteristics: End user perspectives in the kingdom of Saudi Arabia (KSA)," in *Proc. 12th ACM/IEEE Int. Symp. Empirical Softw. Eng. Meas.*, Oulu, Finland, Oct. 2018, pp. 1–10.

[18] S. Aljarallah and R. Lock, "Software sustainability from a user perspective a case study of a developing country (Kingdom of Saudi Arabia)," in *Proc. Int. Conf. Comput., Electron. Commun. Eng. (iCCECE)*, Aug. 2018, pp. 1–6.

[19] P. Bambazek, I. Groher, and N. Seyff, "Requirements engineering for sustainable software systems: A systematic mapping study," *Requirements Eng.*, no. 28, pp. 481–505, Jun. 2023.

[20] V. Bevanda, C. Silveira, and M. Reis, "Sustainability in software engineering: A design science research approach," in *Proc. ERAZ*, Prague, Czech Republic, May 2022, pp. 317–323.

[21] B. Penzenstadler, V. Bauer, C. Calero, and X. Franch, "Sustainability in software engineering: A systematic literature review," in *Proc. 16th Int. Conf. Eval. Assessment Softw. Eng. (EASE)*, Toulon, France, 2012, pp. 1–4.

[22] H. Noman, N. A. Mahoto, S. Bhatti, H. A. Abosaq, M. S. AlReshan, and A. Shaikh, "An exploratory study of software sustainability at early stages of software development," *Sustainability*, vol. 14, no. 14, pp. 1–23, 2022.

[23] E. Kren, M. Dick, S. Naumann, A. Guldner, and T. Johann, "Green software and green IT: An end users perspective," in *Green IT Engineering: Concepts, Models, Complex Systems Architectures*. Heidelberg, Germany: Springer, 2011, pp. 31–54.

[24] B. Penzenstadler, V. Bauer, C. Calero, and X. Franch, "Sustainability in software engineering: A systematic literature review," in *Proc. 16th Int. Conf. Eval. Assessment Softw. Eng. (EASE)*, London, U.K., May 2012, pp. 32–41.

[25] D. Ameller, X. Franch, C. Gómez, S. Martínez-Fernández, J. Araújo, S. Biffl, J. Cabot, V. Cortellessa, D. M. Fernández, A. Moreira, H. Muccini, A. Vallecillo, M. Wimmer, V. Amaral, W. Böhm, H. Bruneliere, L. Burgueño, M. Goulão, S. Teufl, and L. Berardinelli, "Dealing with non-functional requirements in model-driven development: A survey," *IEEE Trans. Softw. Eng.*, vol. 47, no. 4, pp. 818–835, Apr. 2021.

[26] A. Jarzebowicz and P. Weichbroth, "A qualitative study on non-functional requirements in agile software development," *IEEE Access*, vol. 9, pp. 40458–40475, 2021.

[27] M. S. Kumar, A. Harika, C. Sushama, and P. Neelima, "Automated extraction of non-functional requirements from text files: A supervised learning approach," in *Handbook of Intelligent Computing and Optimization for Sustainable Development*, 2022, pp. 149–170.

[28] N. Afreen, A. Khatoon, and M. Sadiq, "A taxonomy of software's non-functional requirements," in *Proc. ICT*, New Delhi, India. New York, NY, USA: Wiley, vol. 1, 2016, pp. 47–53.

[29] P. Lago, S. A. Koçak, I. Crnkovic, and B. Penzenstadler, "Framing sustainability as a property of software quality," *Commun. ACM*, vol. 58, no. 10, pp. 70–78, Sep. 2015.

[30] C. Baker, L. Deng, S. Chakraborty, and J. Dehlinger, "Automatic multi-class non-functional software requirements classification using neural networks," in *Proc. IEEE 43rd Annu. Comput. Softw. Appl. Conf. (COMPSAC)*, Milwaukee, WI, USA, vol. 2, Jul. 2019, pp. 610–615.
[31] B. Li and X. Nong, "Automatically classifying non-functional requirements using deep neural network," *Pattern Recognit.*, vol. 132, Dec. 2022, Art. no. 108948.
[32] S. Kopczyńska, M. Ochodek, and J. Nawrocki, "On importance of non-functional requirements in agile software projects—A survey," in *Integrating Research and Practice in Software Engineering*. Heidelberg, Germany: Springer, 2020, pp. 145–158.
[33] B. Gezici and A. K. Tarhan, "Systematic literature review on software quality for AI-based software," *Empirical Softw. Eng.*, vol. 27, no. 3, 2022, Art. no. 66.
[34] M. W. Suman and M. D. U. Rohtak, "A comparative study of software quality models," *Int. J. Comput. Sci. Inf. Technol.*, vol. 5, no. 4, pp. 5634–5638, 2014.
[35] T. Davuluru, J. Medida, and V. S. K. Reddy, "A study of software quality models," in *Proc. Int. Conf. Adv. Eng. Technol. Res. (ICAETR)*, Unnao, India, Aug. 2014, pp. 1–8.
[36] K. M. Adams, *Non-Functional Requirements in Systems Analysis and Design*, vol. 28. Cham, Switzerland: Springer, 2015.
[37] S. Yadav and B. Kishan, "Analysis and assessment of existing software quality models to predict the reliability of component-based software," *Int. J. Emerg. Trends Eng. Res.*, vol. 8, no. 6, pp. 2824–2840, Jun. 2020.
[38] A. M. Abdullahi, N. K. Yap, A. A. Ghani, H. Zulzalil, N. I. Admodisastro, and A. A. Najafabadi, "A systematic mapping of quality models for AI systems, software and components," *Appl. Sci.*, vol. 12, no. 17, pp. 1–19, 2022.
[39] J. P. Miguel, D. Mauricio, and G. Rodríguez, "A review of software quality models for the evaluation of software products," *Int. J. Softw. Eng. Appl.*, vol. 5, no. 6, pp. 31–53, Nov. 2014.
[40] A. Kaur, "A systematic literature review on empirical analysis of the relationship between code smells and software quality attributes," *Arch. Comput. Methods Eng.*, vol. 27, no. 4, pp. 1267–1296, Sep. 2020.
[41] M. Al Hinai and R. Chitchyan, "Engineering requirements for social sustainability," in *Proc. ICT Sustainability*, Amsterdam, The Netherlands, 2016, pp. 79–88.
[42] D. Kici, G. Malik, M. Cevik, D. Parikh, and A. Basar, "A BERT-based transfer learning approach to text classification on software requirements specifications," in *Proc. 34th Can. Conf. AI*, Vancouver, BC, Canada, 2021, pp. 1–13.
[43] T. Hey, J. Keim, A. Koziolek, and W. F. Tichy, "NoRBERT: Transfer learning for requirements classification," in *Proc. IEEE 28th Int. Requirements Eng. Conf. (RE)*, Zurich, Switzerland, Aug. 2020, pp. 169–179.
[44] D. St-Louis and W. Suryn, "Enhancing ISO/IEC 25021 quality measure elements for wider application within ISO 25000 series," in *Proc. 38th Annu. Conf. IEEE Ind. Electron. Soc. (IECON)*, Montreal, QC, Canada, Oct. 2012, pp. 3120–3125.
[45] M. Binkhonain and L. Zhao, "WITHDRAWN: A review of machine learning algorithms for identification and classification of non-functional requirements," *Expert Syst. Appl.*, vol. 1, Feb. 2019, Art. no. 1000001.
[46] A. Rashwan, O. Ormandjieva, and R. Witte, "Ontology-based classification of non-functional requirements in software specifications: A new corpus and SVM-based classifier," in *Proc. IEEE 37th Annu. Comput. Softw. Appl. Conf.*, Washington, DC, USA, Jul. 2013, pp. 381–386.
[47] J. Ding, Y. Li, H. Ni, and Z. Yang, "Generative text summary based on enhanced semantic attention and gain-benefit gate," *IEEE Access*, vol. 8, no. 1, pp. 92659–92668, 2020.
[48] S. Bird, E. Klein, and E. Loper, *Natural Language Processing With Python: Analyzing Text With the Natural Language Toolkit*. Sebastopol, CA, USA: O'Reilly Media, 2009.
[49] M. Honnibal and I. Montani, "spaCy 2: Natural language understanding with Bloom embeddings," *Convolutional Neural Netw. Incremental Parsing*, vol. 7, no. 1, pp. 411–420, 2017.
[50] M. Lima, V. Valle, E. Costa, F. Lira, and B. Gadelha, "Software engineering repositories: Expanding the PROMISE database," in *Proc. 33rd Brazilian Symp. Softw. Eng.*, Salvador, Brazil, Sep. 2019, pp. 427–436.
[51] T. Iqbal, P. Elahidoost, and L. Lúcio, "A bird's eye view on requirements engineering and machine learning," in *Proc. 25th Asia–Pacific Softw. Eng. Conf. (APSEC)*, Nara, Japan, Dec. 2018, pp. 11–20.
[52] Z. S. H. Abad, O. Karras, P. Ghazi, M. Glinz, G. Ruhe, and K. Schneider, "What works better? A study of classifying requirements," in *Proc. IEEE 25th Int. Requirements Eng. Conf. (RE)*, Lisbon, Portugal, Sep. 2017, pp. 496–501.
[53] A. Casamayor, D. Godoy, and M. Campo, "Semi-supervised classification of non-functional requirements: An empirical analysis," *Inteligencia Artif., Revista Iberoamericana de Inteligencia Artificial*, vol. 13, no. 44, pp. 35–44, Jan. 2010.
[54] R. Navarro-Almanza, R. Juarez-Ramirez, and G. Licea, "Towards supporting software engineering using deep learning: A case of software requirements classification," in *Proc. 5th Int. Conf. Softw. Eng. Res. Innov. (CONISOFT)*, Mérida, Mexico, Oct. 2017, pp. 116–120.
[55] Z. Kurtanovic and W. Maalej, "Automatically classifying functional and non-functional requirements using supervised machine learning," in *Proc. IEEE 25th Int. Requirements Eng. Conf. (RE)*, Lisbon, Portugal, Sep. 2017, pp. 490–495.
[56] R. Jindal, R. Malhotra, and A. Jain, "Automated classification of security requirements," in *Proc. Int. Conf. Adv. Comput., Commun. Informat. (ICACCI)*, Jaipur, India, Sep. 2016, pp. 2027–2033.
[57] A. T. Peterson, S. Knapp, R. Guralnick, J. Soberón, and M. T. Holder, "The importance of data quality for generating reliable distribution models for rare, elusive, and cryptic species," *PLoS ONE*, vol. 4, no. 6, 2019.
[58] N. Condori-Fernandez and P. Lago, "Characterizing the contribution of quality requirements to software sustainability," *J. Syst. Softw.*, vol. 137, pp. 289–305, Mar. 2018.
[59] X. Chen, N. Hong, A. Choudhary, R. White, and J. Rudi, "The importance of software sustainability in open source computational science and engineering," *Comput. Sci. Eng.*, vol. 6, no. 1, pp. 21–28, 2018.
[60] C. C. Venters, C. Jay, L. M. Lau, M. K. Griffiths, V. Holmes, R. R. Ward, J. Austin, C. E. Dibsdale, and J. Xu, "Software sustainability: The modern tower of Babel," in *Proc. 3rd RESuSy*, Karlskrona, Sweden, 2014, pp. 7–12.
[61] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," 2018, *arXiv:1810.04805*.
[62] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. U. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 5998–6008.
[63] J. G. Gaudreault, P. Branco, and J. Gama, "An analysis of performance metrics for imbalanced classification," in *Proc. ICDS*, Halifax, NS, Canada. Cham, Switzerland: Springer, 2021, pp. 67–77.
[64] I. Ul Hassan, R. H. Ali, Z. Ul Abideen, T. A. Khan, and R. Kouatly, "Significance of machine learning for detection of malicious websites on an unbalanced dataset," *Digital*, vol. 2, no. 4, pp. 501–509, 2022.

AHMAD F. SUBAHI received the B.Sc. degree in computer science from King Abdulaziz University (KAU), Jeddah, Saudi Arabia, in 2002, the M.Sc. degree in information technology form the Queensland University of Technology (QUT), Brisbane, Australia, in 2008, and the M.Sc. degree in advanced computer science and the Ph.D. degree in computer science from the University of Sheffield, U.K., in 2010 and 2015, respectively. He is currently an Associate Professor in software engineering with the Computer Science Department, University College of Al Jamoum (JUC), Umm Al-Qura University (UQU), Makkah, Saudi Arabia. His research interests include automated software development, model-driven engineering, domain specific (modeling) languages, model transformations, code generation and programming languages design, software systems architecture and design, (graph) database systems, the secure IoT systems engineering, and the application of AI/ NLP techniques in software engineering domain.