MANCHESTER
1824
The University of Manchester

Classification of User Stories for Software Development

A dissertation submitted to The University of Manchester for the degree of
Master of Science in Data Science (Computer Science Data Informatics)
in the Faculty of Humanities

Year of submission
2023

Student ID
11166812

School of Social Science

Contents

Word count: 10017

List of Figures

List of Tables

Abstract

Context: The use of user stories as artefacts to record user requirements for software development is widely acknowledged. They consist of concise texts with a semi-structured scheme that uses natural language to explain requirements. Qualitative analysis and categorisation of user stories before assigning them to development teams avoid potential issues at the later stage of the software development life cycle. While many approaches have been applied to classify user stories, they fail to provide adequate performance due to the inherent nature of natural language and the scarcity of labelled data.

Objective: This study addresses these problems by using a zero-shot learning-based model for data labelling. After data labelling, we fine-tune transformer-based language models over the labelled dataset for classifying user stories into pre-defined class labels.

Methods: The data labelling approach uses the ***sileod/deberta-v3-large-tasksource-nli*** model with self-training. For classification, we fine-tune pre-trained models like *BERT, ALBERT, DistilBERT, RoBERTa, XLNet, GPT-2* and *BERT4RE* over the given user story dataset.

Results: The study shows that the zero-shot model achieves a state-of-the-art F1 score of **93%** over the test dataset. The ***RoBERTa*** model achieves the best F1 score close to **80%**. The BERT-based models performed well over the GPT-based and domain-specific *BERT4RE* model. The macro-averaged performance metrics are used for calculating the F1 score.

Conclusion: The study demonstrates a comparative analysis of language models using performance metrics and benchmarking for user story classification tasks. It also showcases the use of zero-shot models using self-training for dataset labelling which achieves industry-leading performance thus mitigating the labelled data shortage problem in requirements classification.

Chapter 1

Introduction

1.1 Context and Motivation

The software development (SD) for any project adheres to a set of requirements from various stakeholders (both technical and non-technical). These requirements are documented in natural language from the user's point of view [1]. The requirements may be represented formally in the form of software requirements specification (SRS) or in the form of user stories (US) which are easily understandable by the stakeholders. The US serve as a high-level, semi-structured natural language expression of requirements [2]. The semi-structured template is often in the form of *As a <TYPE of user>, I want <SOME goal> so that <SOME reason>.* [3]. A US's goal is to present how a software component will benefit the user. In Scrum, user stories are being used more frequently to describe requirements [4].

Natural language processing (NLP) has significantly advanced in recent years, especially with the introduction of transformer-based language models (TBLM) [5]. The TBLM have achieved new benchmarks in a variety of NLP tasks, including BERT [6], GPT-2 [7], and RoBERTa [8]. These models have the power to completely alter a range of applications, including sentiment analysis and machine translation (cite:vaswani2017attention). Although these models provide encouraging results, there is an increasing need to comprehend their behaviour, advantages, and disadvantages in practical situations, particularly for specialised tasks like user story classification. By ensuring that stakeholders and developers have the same knowledge and priorities, effective user story classification can greatly improve the software development process. It is essential to investigate their capabilities in the context of user story categorization given the potential of transformer-based models in comprehending and producing human-like language.

1.2 Aims and Objectives

This study aims to evaluate different TBLMs on a given US dataset by dataset labelling as per the six pre-defined classes (*Chat, Contacts, Aggregation, Accessibility, Consistency and Authentication*) and address the research gaps discussed in the next chapter. The US dataset is collected from two sources consisting of both labelled and unlabeled data rows. This study also provides an end-to-end mechanism for dataset validation, labelling, classification, model selection and benchmarking over the given dataset.

The following objectives are used to reach the main goal of our project:-

- Overview of the related studies and models currently used for the US classification.
- Dataset validation and anonymisation for qualitative and ethically approved analysis over the collected US dataset.
- Comparing dataset labelling semi-supervised techniques used for annoying the given US dataset.
- Finetuning Bert-based and GPT-based language models for precise multi-class classification of the US into predefined class labels.

By achieving these goals, this project hopes to offer practitioners a comprehensive process that streamlines the annotation and categorization of the US, thereby making it simpler for them to determine and group requirements in SD.

1.3 Intended Contributions

The research looks at computational efficiency, a crucial aspect of modern software development, in addition to the spectrum of classification capabilities. The study thoroughly assesses temporal aspects, including the amount of time required for the selected model's training and inferences. Significant efforts have been made to investigate computing efficiency, which has a direct bearing on the viability and scalability of incorporating US classification during the SD process. Therefore the study addresses the research gaps and answers the following research questions:-

RQ-1: What is the level of performance of semi-supervised and zero-shot learning using self-training for labelling the given user-story dataset?

RQ-2: How do various fine-tuned transformer-based language models compare in terms of their performance for classifying the given user-story data?

RQ-3: How does the choice of the evaluation metrics impact the understanding and assessment of transformer-based language model performance in classifying user stories?

RQ-4: How does the domain-specific model perform in comparison with the generic models?

Aside from this introduction chapter, this study is separated into five sections. We go over the background concepts needed for further investigation and related work in Chapter 2, along with any pertinent research gaps. The study's methodologies are described in depth in Chapter 3 and the study's results are examined in Chapter 4. Chapter 5 evaluates the findings after which there is a detailed discussion of additional restrictions and threats to the study's validity. Chapter 6 brings the study to a conclusion and addresses potential directions for further research.

Chapter 2

Background and related work

2.1 Background

2.1.1 Agile Software Development and User Stories

Agile is a broad term for a set of software development principles. It is a theoretical framework for software development that starts with the planning stage and progresses through incremental and iterative interactions throughout a project's life cycle until the deployment stage. With the capacity to embrace changes without endangering the process or requiring unnecessary rework, the initial goal of agile approaches is to lower the inefficiency in the process of developing software [9].

Agile methodologies come in many forms, including Test-Driven Development (TDD), Feature Development (FDD), Extreme Programming (XP), Scrum, Dynamic System Development Model (DSDM), Crystal, and others. Each technique has its own guidelines, life cycle, roles, benefits, and drawbacks, among other things. All these different agile software development techniques construct the software through incremental and iterative procedures [10].

The US is widely used in the SD domain for conveying requirements by the end user. The US is composed of natural language and hence understood by technical and non-technical stakeholders. They have gained significant acceptance in agile SD [11]. The US create an agreement of what the end-user anticipates from the product within a team by encouraging all stakeholders to consider and discuss the specifics of a demand. The suitable software can be developed thanks to this evaluation of the proper criteria. This reduces costly rework, boosts productivity, raises the calibre of output and lowers the cost of software development [12].

US contain three key components *WHO, WHAT and WHY*. These elements are used for validating the structure and quality of the US. Table 2.1 contains the examples of US used in SD and the relevant extracted key elements.

WHO: defines a stakeholder, user or actor who wants the functionality.

WHAT: defines the task/component/module/change/build that is required want.

WHY: defines the reason and sometimes a concise background of the task which is often optional but is recommended for forging high-quality US [1].

User Story	**Who**	**What**	**Why**
As a user, I want to register an account so that I can access premium features.	User	Register an account	Access premium features
As a developer, I want a version control system so that I can track and manage changes to the codebase.	Developer	Use a version control system	Track and manage code changes
As an admin, I want to view user activity logs so that I can monitor and ensure system security.	Admin	View user activity logs	Monitor system security
As a project manager, I want to generate monthly reports so that I can review team progress.	Project Manager	Generate monthly reports	Review team progress
As a tester, I want to report bugs so that they can be fixed before the software is released.	Tester	Report bugs	Ensure software quality

Table 2.1. User-stories and their relevant key elements

Apart from the structure, the US must also follow the INVEST guidelines which define that a user story must be *Independent, Negotiable, Valuable, Estimable, Small and Testable* [13]. A complete and unambiguous user story is constructed if it has a clear role, goal (either soft or hard), task and/or capability [3].

The six custom classes are described below based on various aspects and stakeholders present at different stages of SD.

Authentication: Centred around confirming the identities of users, devices, or systems. The main participants in these stories are the end-users, system administrators, and security teams. The stories encapsulate elements such as user login, registration, password resetting, multi-factor authentication, and session management.

Accessibility: Ensures the software is user-friendly for individuals with varying abilities and disabilities. The key players here are the end-users, UX/UI designers, and compliance teams. The stories address components like compatibility with screen readers, colour contrast, keyboard navigation, and adherence to accessibility standards.

Aggregation: Concerned with gathering and presenting data from diverse sources in a coherent and meaningful manner. The primary stakeholders are the end-users, data analysts, and business intelligence teams. These stories encompass aspects like data collection, data processing, data visualisation, and generation of reports.

Chat: Relates to the real-time communication features within the application. The main participants in these stories are the end-users, customer service teams, and community managers. The stories could involve tasks such as creating chat rooms, sending and receiving messages, managing chat history, and implementing notifications.

Contacts: Concerned with the management of user contacts within the application. The key players here are the end-users, sales teams, and customer relationship management teams. These stories address components like adding, editing, and deleting contacts, integrating with external contact lists, and managing contact groups.

Consistency: Ensures that the application's design, behaviour, and interactions are consistent across different parts of the application. The primary stakeholders are the end-users, user experience/interface designers, and product managers. These stories encompass aspects like maintaining a consistent look and feel throughout the application, consistent error messages, and consistent user interface behaviour.

2.1.2 Natural Language Processing and Text Classification

Natural language processing (NLP) automates the analysis, modelling, inferring and representation of human language. It is usually incorporated for text preprocessing. NLP techniques are often used to increase the quality of US. Considering the application of NLP it is often applied at the initial stages of software development and requirements engineering like extracting key components of a US, entity extraction, ambiguity detection, functional and non-functional requirements categorisation [14].

The following NLP pipeline steps are applied for text preprocessing:

Tokenization: Text is divided into tokens, which can be phrases, symbols, words, or other significant elements. This step's primary objective is to examine each word in a sentence. A parser that handles the tokenization of the documents is necessary for text categorisation as well as text mining [15].

Stop-Words: Many words that are employed in text and classification of documents do not have a significant enough meaning to be utilised by classification algorithms. Eliminating these words from texts and documents is the method used to deal with them most frequently.

Lowercase: Texts use a variety of caps. Multiple capitalisations can be quite difficult to categorise lengthy paragraphs because they sometimes contain several phrases. To deal with irregular capitalisation, texts are usually converted to lowercase.

Contractions: Expanding contractions help to standardise language since they are words or groups of words that are abbreviated by omitting letters and substituting them with apostrophes.

Punctuation: The majority of text and document data sets include a lot of extra punctuation and special characters. Special characters and critical punctuation are necessary for humans to understand documents, but they impact classification algorithms.

Stemming and Lemmatization: The technique of stemming involves reducing words into their root form. Even if the stem is not a dictionary word, the goal of stemming is to reduce similar words to the same stem. Snowball Stemmer is usually effective [16]. Lemmatization, as opposed to stemming, reduces words to their root word, appropriately reducing the target word, and making sure that the root word is a part of the language.

Parts-Of-Speech (POS) Tagging: POS tagging is the process of identifying the part of speech that each of the words in a sentence uses. The main goal of POS labelling is to determine a word's grammatical group, such as whether it belongs to a noun(s), pronoun(s), adjective, verb, or adverb, depending on the context. POS tagging helps in lemmatization for instance if the word ”leaves” is to lemmatized then its equivalent is ”leaf” but if the word was tagged as *verb* then the lemmatized equivalent word is ”leave”. Using POS tagging the accuracy is improved [17].

Text classification (TC) is the process of categorising a collection of textual information into predetermined groups. To accomplish classification, a data set for training with many examples of data input information along with corresponding labels is required. The best technique to associate training data set samples with particular class labels will be decided by the machine learning model. Features are extracted, dimensions are reduced, classifiers are chosen, and evaluation is done and the same is used at different levels of scope, such as document, paragraph, sentence, and sub-sentence levels. Based on the number of classes present TC can be categorised into binary, multi-class and multi-label classification [18].

Classification of labelled data can be of three types for instance binary, multi-class and multi-label classification. In binary classification, there are two distinct labels and all the data rows belong to either of them but not both the classes. Predicting zero or more class labels that are mutually non-exclusive (*Events that can occur simultaneously together are known as mutually non-exclusive events in probability*) is known as multi-label classification. A single observation might therefore belong to multiple class labels in multi-label categorization. On the other hand, multi-class classification involves the prediction of zero or more class labels that are mutually exclusive (*Events that cannot coexist are said to be mutually exclusive in probability*). This means that just one category may contain all of the observations made. The multi-class classification is used in our project for the classification of each US to a single class [19].

2.1.3 Machine Learning

Machine Learning (ML), a subsection of artificial intelligence focuses on creating statistical models and algorithms that let computers carry out tasks without explicitly programming them. These systems have strong decision-making abilities as they gain knowledge and develop via experience. They are frequently utilised in numerous practical applications, such as virus detection, spam filtering, face and object identification, speech recognition, and more [20]. The four categories of machine learning are supervised, unsupervised, semi-supervised and reinforced learning.

Supervised ML: Utilising labelled data, supervised machine learning approaches can predict future events by utilising what the model has already learned in the past. The learning technique creates an inference function to predict the outputs based on an analysis of a recognised training data set [21]. The predictions done by the models are then compared against the actual labels. The mismatch between the predicted and true labels is the error that the algorithm tries to minimise. For example, the logistic regression algorithm is often used for functional requirements classification tasks [22].

Semi-Supervised ML: Utilises data with and without labels to carry out specific learning tasks. Combines unsupervised with supervised learning, making it possible to employ the enormous volumes of unlabelled data often present in many use cases and the smaller groups of labelled data for further analysis. Semi-supervised learning (SSL) is especially pertinent in situations with a dearth of labelled data, making the development of trustworthy supervised learning difficult [23]. Semi-supervised learning uses techniques like self-training (*which we have used in our study to label our user-story dataset*) [24], co-training [25] and graph-based label propagation [26].

Unsupervised ML: Involves the recognition of patterns without the use of a target feature. The strategies are accepted as clustering and association data mining since all the variables utilised in the study are used as inputs. They are appropriate for adding labels to the data that will later be utilised to carry out supervised learning activities [27]. Unsupervised clustering algorithms find natural groupings in the unlabelled data and then give each data value a label [28]. A recent study utilises the K-means clustering algorithm for the US classification task [29] which groups similar Tf-Idf vector patterns in a cluster.

Reinforced ML: Reinforcement engages with its environment by taking actions and identifying successes or rewards. Trial-and-error learning and delayed rewards are the two most crucial components of reinforcement learning. This method aids computer programs and software systems in automatically determining the best course of action in a specific situation to enhance the performance of the machine learning model [30]. The reward function selected and the environment's layout determine the calibre of the outcomes produced by this learning. For example, the reinforcement learning technique is used for requirements traceability tasks [31].

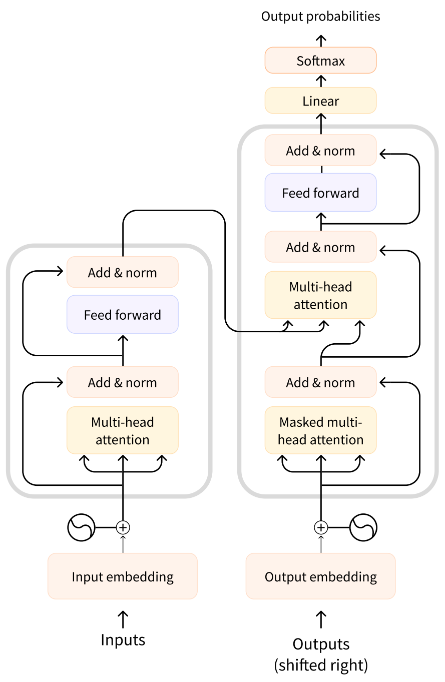

Fig. 2.1. Transformer Model Architecture [34]

2.1.4 Transformer-based Language Models

The Recurrent Neural Networks (RNNs) were majorly used in NLP applications. This made these models very handy for classification and regression tasks. RNNs removed the need for re-training for different domains by generalisation. Two architectures played a major role: Long Short Term Memory (LSTM) [32] and Gated Recurrent Unit (GRU) [33]. However, these architectures were hard to do for multi-processing computation and long sequences of input results in vanishing gradient problems. The TBLM architecture proposed a new attention architecture [34].

The Transformer is built on a system of encoders and decoders that converts a sequence of characters called X into a latent representation called Z that is equal to (z_1,..., z_n). As a result of this model's auto-regressive feature, each element of the output sequence Y_m=(y_1,..., y_m) is produced separately. In other words, the sequence Y_{m-1}=(y_1,..., y_{m-1}) and the latent representation of Z were used to form the word Y_m. The Multi-Head Attention layer is utilised by both the encoder and decoder. The single Attention layer converts the weighted sum of each value V from a query Q and keys K.

The transformer-based models are broadly categorised into three categories:-

Encoder Models: The transformer architecture's encoder component is used in these models. They typically perform the input sequence processing all at once and are employed for applications like text categorisation, analysing sentiment, recognition of entities, etc. Bidirectional Encoder Representations from Transformers, or BERT, is an illustration of an encoder model.

Decoder Models: These models make use of the transformer architecture's decoder component. They are frequently employed for jobs like text production, translation, etc. since they produce the output sequence step-by-step. A decoder model would be something like the GPT (Generative Pre-trained Transformer).

Encoder-Decoder Models (or Seq2Seq Models): Both the encoder and decoder of the transformer are used in these versions. They use the encoder for analysing an input sequence and the decoder to produce an output sequence. They frequently perform tasks like summarising and machine translation. The T5 (Text-to-Text Transfer Transformer) model is an illustration of an encoder-decoder.

Bidirectional Encoder Representations from Transformers: BERT model's architecture comprises multiple layers of bidirectional transformer encoders. Unlike traditional language models pre-trained using left-to-right and vice-versa methods, the BERT is pre-trained using two unsupervised methods defined as masked language models (MLM) and next sentence prediction (NSP) [6]. BERT is trained upon the BookCorpus with 800M words and the English Wikipedia with 2,500M words. Fine-tuning of BERT is relatively more efficient than pre-training. BERT model types have been illustrated in Table 2.2 and further, these models can be divided into cased, uncased and multilingual.

Model	Hidden Layers (L)	Hidden Size (H)	Attention Heads (A)	Total Parameters
BERT-base	12	768	12	110 Million
BERT-large	24	1024	16	340 Million
DistilBERT	6	768	12	66 Million
RoBERTa-base	12	768	12	125 Million
RoBERTa-large	24	1024	16	355 Million
ALBERT-base	12	768	12	11 Million
ALBERT-large	24	1024	16	18 Million
DeBERTa-base	12	768	12	110 Million
DeBERTa-large	24	1024	16	340 Million

Table 2.2. Comparison of BERT and BERT-like Models

Hidden Layers (L): This is the number of transformer blocks in the model.

Hidden Size (H): This is the size of the representation that is learned by the model.

Attention Heads (A): This is the number of attention heads in the transformer blocks.

Total Parameters: This is the total number of parameters in the model.

DistilBERT is a smaller version of BERT that retains approximately ninety-five of BERT's performance while being sixty per cent smaller and sixty per cent faster [35].

RoBERTa is a variant of BERT that uses a different training approach and removes the next sentence prediction task, leading to improved performance [36].

ALBERT is another variant of BERT that uses factorised embedding parameterization and cross-layer parameter sharing to significantly reduce the number of parameters [37].

DeBERTa (Decoding-enhanced BERT with disentangled attention) improves the BERT and RoBERTa models using two techniques: disentangled attention and an enhanced mask decoder [38].

Generative Pre-trained Transformer: The GPT language model makes use of transfer learning and unsupervised learning techniques. It is initially trained on a big corpus of text without labels before being tailored for particular tasks employing supervised learning. Two stages of training are used for training the GPT model. Unsupervised pre-training is used in the first stage to learn the model's foundational parameters from an extensive database of unlabelled text. In the second stage, known as supervised fine-tuning, these parameters are adjusted to a particular job by applying the relevant supervised objective [7].

Model	**Hidden Layers (L)**	**Hidden Size (H)**	**Attention Heads (A)**	**Total Parameters**
GPT-1	12	768	12	117 Million
GPT-2	12/24/36/48	768/1024/1280/1600	12/16/20/25	117M/345M/774M/1.5B
GPT-3	8/12/24/48	768/1024/1280/1600	12/16/20/25	125M/350M/760M/175B
GPT-NeoX	44	6144	64	20 Billion
GPT-4**	96	16384	128	345 Billion

Table 2.3. Comparison of GPT and GPT-like Models

GPT-NeoX is mainly based on the GPT-3 model architecture. Instead of learnt positional embeddings, which are utilised in GPT models, GPT-NeoX uses rotary embedding. In contrast to processing the attention and feed-forward layers sequentially, they are computed in parallel and summed [39].

2.2 Related Work

2.2.1 Existing Approaches

US are written from a user's perspective which makes them semi-structured and ambiguous. Classification of the US is a common practice in SD. A handful of research is available in US classification but no research relates directly to our work of the end-to-end process of annotating and then classifying US based on pre-defined classes.

In the recent research [40], the authors discuss the issues with SRC, which entails determining which category a particular Software Requirement (SR) belongs to. Even though software requirements are well known and extensively characterised, automatic classification of requirements stated in plain language into functional requirements and non-functional requirement subcategories remains a challenge. This occurs because stakeholders and requirements engineers use various terminologies and sentence structures to convey the same type of requirement.

Study conducted by Bhawnesh Kumar [29] suggests a method of clustering the US using the k-means algorithm [41]. Similar US are grouped together using this method based on similarity metrics. The quality of the resulting cluster improves as the value of *k* increases, according to the findings of their experiments. The method uses the term frequency and inverse term frequency (TF-IDF) [42], which is superior to the count vectorizer in terms of producing balanced clusters. The total software development life cycle is shortened by this method, which also minimises the time it takes to implement requirements. A case study is used to illustrate our process while taking into account various US in their study. The relevance of the work done by the authors indicate that the US can be grouped into similar clusters based on the TF-IDF vectors.

The authors [43] presented an autonomous machine learning strategy to categorise user stories based on crucial components in a user narrative, such as user type, the function of the US, and the entity the function is executed. They preprocessed the US into n-grams and then used a TF-IDF word embedding model to construct vectors that are input into a Naive-Based Classifier to categorise future incoming user tales into a labelled category. The experiment findings show that the vectors created from the word embeddings played an important role in the classifier correctly classifying the user story.

Previous research on the classification of requirements done by Kurtanovic et al. [44], focuses on classifying requirements into FR and NFR using the Support Vector Classifier. The categorisation was accomplished using binary and multi-class classification using lexical contexts. The classes include performance, usability, operations and security, The authors' obtained precision and recall for automatically recognising FRs and NFRs, as well as identifying individual NFRs, were up to 90 per cent.

Regarding issues of requirement data scarcity, the authors [45] suggested using a ZSL-based approach with contextual word embeddings and TBLM like Sentence-BERT (generic model), Bert4RE and BertOverflow (domain-specific models) available on the Hugging-Face library [46]. The generic models performed better than the domain-specific models and the simple label strategy was better than word embedding-based ones. They achieved an F1 score of 66 per cent for FR/NFR classifications and frequent classes the score was between 72 to 80 per cent. We have built on a similar approach for data annotation and managed to increase the ZSL F1 score by applying self-training using a small (1 to 3 per cent of the entire dataset) labelled dataset [47].

Similarly, the research conducted on semi-supervised learning by Croce, D [48] uses the capability of Generative Adversarial Networks (GANs) together with BERT-like architectures to provide robust text categorisation with a small number of labelled cases. The experimental results show that GAN-BERT can dramatically reduce the number of annotated examples required (down to 50-100) while still performing well on various phrase categorisation tasks.

2.2.2 Research Gaps and Dissertation Contributions

According to our literature review, the previous studies employed Classification Algorithms and Word Embedding Techniques to automatically classify requirements according to their functionality. Existing methods classify needs using semantic homology and language heuristics. There is substantially less research focused on employing Transformer-based models to classify the US. The research gaps identified are listed below, together with my intended research contributions:

- Kochbati [49], applied the TF-IDF embeddings with the Hierarchical Agglomerative Clustering algorithm (HAC) to categorise the requirements. The same workflow is also followed in the research done by Bhawnesh Kumar [29] using k-means clustering algorithm instead of HAC. In our study we extend these researches by utilising TBLM which has better performance and handles out-of-vocabulary words with semantic similarity.
- In [40], a Naive-biased Classifier was used to categorise the requirements according to their functionality. This study can be expanded by using TBLM like BERT and GPT for requirement categorisation, comparing word embedding models, and choosing the model with the highest accuracy using performance measures.
- Using the ZSL-based approach used in [45] can be extended using small labelled data (around ten examples per class) to achieve high accuracy by applying self-training.

Including secondary studies for summarising related work for the role of NLP in the US as aggregated by the authors in [14]. Their analysis showcases the use of NLP techniques in various requirements engineering tasks such as defect identification, generating software artefacts, expanding abstractions and establishing links between requirements. Their system literature review can be extended for use cases where TBLM are applied. Table 2.4 summarises both primary and secondary works.

Reference	**Category**	**Goal**
D. Croce [48] and W. Alhoshan [45]	User-story labelling	Applying semi-supervised and ZSL approaches for dataset labelling.
M. Jurisch [43]		Autonomous categorisation based on key elements present in the US.
Z. Kurtanovi [44]	User-story classification	Binary and multi-class classification using lexical contexts.
M. Jurisch [43]		Autonomous categorisation based on key elements present in the US.
F. Dalpiaz [50]	Identifying defects	Identifying near-synonyms. Applied semantic-similarity and extraction of conceptual models NLP techniques.
G. Lucassen [12] and M. Galster [51]		Identification and assessing of quality attributes in the US.
R. Barbosa [52]	Identifying key abstractions	Assessing semantic network connections present in the US.
G. Lucassen [53] and Y. Wautelet [54]		Extracting conceptual models from the US.

Table 2.4. Summary: Related work

Chapter 3

Methodology

To accomplish the study's objectives we perform a series of tasks. We start with identifying and evaluating the related work already accomplished in the field of requirements classification. By doing so we want to make sure that we are not reinventing the wheel but positively contributing towards the SD field. After evaluation of the literature, we move to the data ingestion part where we have two data sources one of which has labelled data and is about 10% of the whole dataset and the other one is an unlabelled dataset. We use semi-supervised techniques using the transformer-based models to annotate the dataset. The resulting labelled dataset is input for the next stage of fine-tuning the TBLMs. The TBLMs are evaluated based on the performance metrics and benchmark scores. The measure of both time and space complexity is vital in the SD process. The resulting model is then used for final classification and saved on disk for further inferences. This marks the end of our research design and the same is illustrated in Fig 3.1. We discuss all the stages with their inputs and outputs in detail in this chapter.

3.1 Dataset Collection and Anonymisation

The data required for our task has been collected from two data sources. The first one is contributed by researchers as a new issue with their relevant labels in a GitHub repository [55]. The second data source by the author Fabiano [56] is a compilation of 22 data sets with 50+ requirements, described as US. These were all available online and licensed for further analysis by the software makers. The GitHub dataset is labelled with the classes that are discussed in Chapter 2. The data was collected using methods implemented using Python version 3.10.11 and the data is stored in Pandas version 2.0.3 data-frame (DF) [57] for further processing and analysis.

Extracting data from [55], a GitHub-issue extractor method is implemented. There are 63 unique labels present including the six classes of our concern. The method filters the classes and only draws the relevant class-labelled data rows. The data often has multiple labels which are manu-

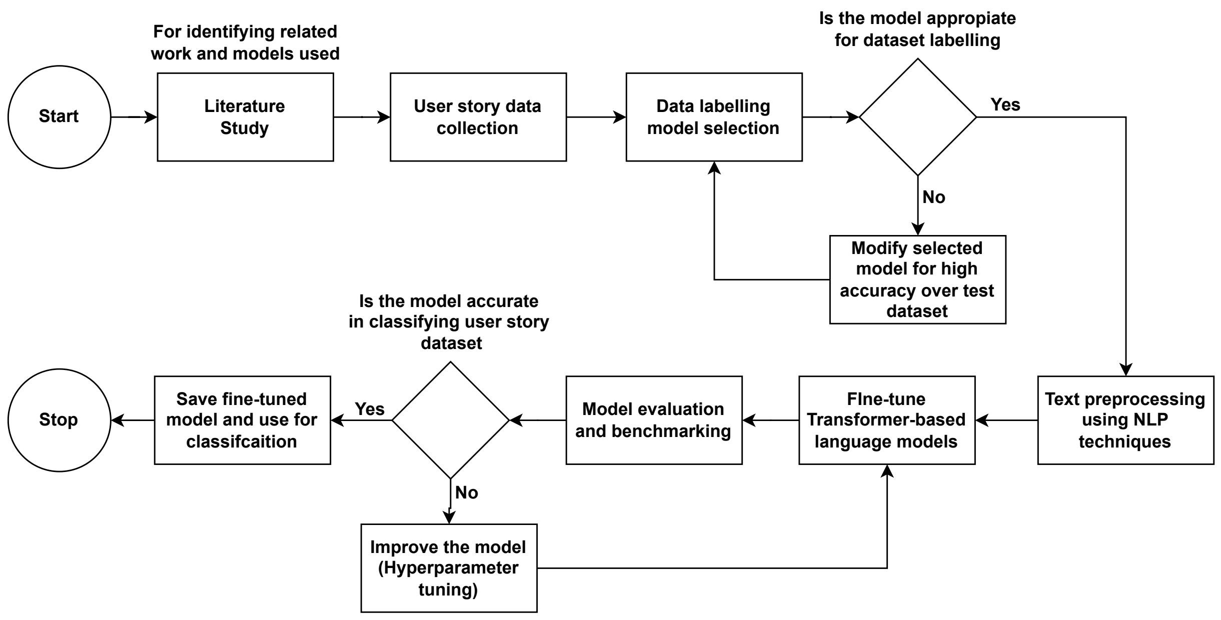
For identifying related work and models used
Start
Literature Study
User story data collection
Data labelling model selection
Is the model appropiate for dataset labelling
Yes
No
Modify selected model for high accuracy over test dataset
Text preprocessing using NLP techniques
FIne-tune Transformer-based language models
Model evaluation and benchmarking
Is the model accurate in classifying user story dataset
Yes
No
Improve the model (Hyperparameter tuning)
Save fine-tuned model and use for classifcaition
Stop

Fig. 3.1. An illustration of the research design structure.

ally selected as per the domain-specific knowledge. The issues that are pending for labelling have been also filtered out of the dataset collection task. The extraction method has been implemented with current Git-Hash matching to flag any update made on the website for the related dataset rows. The second unlabelled data source [56] contains 22 text files. The US is separated by a new-line character. A method is used to read all the lines present in the files and convert them to a DF. The label value is set to *Unknown* for all the data rows present in the *label* column of the DF. After the data ingestion pipeline, the final combined DFs have **2152** data rows and 2 columns *text* and *label*. Both the raw and combined dataset is stored in the dissertation GitHub repository [58] under the *artefacts* directory.

The data collected is publicly available on respective websites with appropriate licenses. The data has not been pre-processed and therefore is not anonymised. The data thus collected needs to be passed through an anonymisation pipeline. Named entity recognition (NER) is used to replace identifiable information like person, organisation, location, phone and email details. As per [59], *SpaCy* NLP library gives better results in identifying and linking NERs to their relevant classes. We used *PERSON*, *ORGANISATION*, *LOCATION*, *PHONE* and *EMAIL* entity types. Each entity is anonymised uniformly across the dataset records (i.e. if ”Steve” is changed to ”Person1” then it needs to be Person1 among all the dataset rows). The anonymised data is henceforth used for further analysis and is overwritten on our dataset *text* column.

3.2 Dataset Preprocessing and Analysis

The US dataset collected needs to be validated on their structure and the INVEST [13] features. We validate our US dataset using the fine-tuned BERT-base model with 97% accuracy for identification of the US in issue records [60]. After applying validation there are 2087 US left which indicates that the aggregated dataset is of high quality as per the standards. The top five filtered-out invalid user stories as per the structure or INVEST features are outlined in Table 3.1.

User Story	**Invalidation Type**
I want to be able to select a specific period to see the "data".	Structure
I want to save the environmental measures of every moment coupled with that specific time so that I can look into them at a later moment.	Structure
As a Solid user, I want to know when I am a Solid user and how to be in control and actively consent.	INVEST
As a User, I want to be able to see some metrics on the use of the game, so that I can see how much it is being used.	INVEST
As UHOPE, I want to have at least one backup at all times of every piece of information in the entire system, so that the data is always available even if something happens to the system.	INVEST

Table 3.1. Top five invalid user stories removed from the dataset.

There are 77 labelled and 2010 unlabelled data rows. Table 3.2 aggregates the labelled data with the relevant class-label names. It should be noted that the label's distribution depicted in Table 3.2 is from the GitHub source only and does not include the unlabelled data rows.

Label	**Count**
Chat	18
Contacts	16
Aggregation	11
Accessibility	11
Consistency	11
Authentication	10

Table 3.2. Distribution of labels from GitHub data source [55].

The TBLM tokenizer has pre-defined *max-length* parameter value (i.e. 128, 256 or 512). We apply an exploratory data analysis technique over the distribution of the US text length depicted in Fig 3.2. The distribution illustrates that the data is left-skewed and the outlier data rows need to be dropped. To accomplish this the difference among the first and third quartiles is computed to calculate the inter-quartile range (IQR), which we then use to locate outliers in the continuously distributed US text length data. Fig 3.3 depicts the distribution after applying the IQR method to the dataset. As per the Fig 3.3 the maximum string length is not more than 256 and hence the *max-length* parameter value for TBLM tokenizer is set to 256.

3.3 Dataset Labelling

For answering RQ-1 the unlabelled dataset needs to be classified for training, evaluating and testing the TBLMs for text classification. We need a model that performs on the test labelled dataset

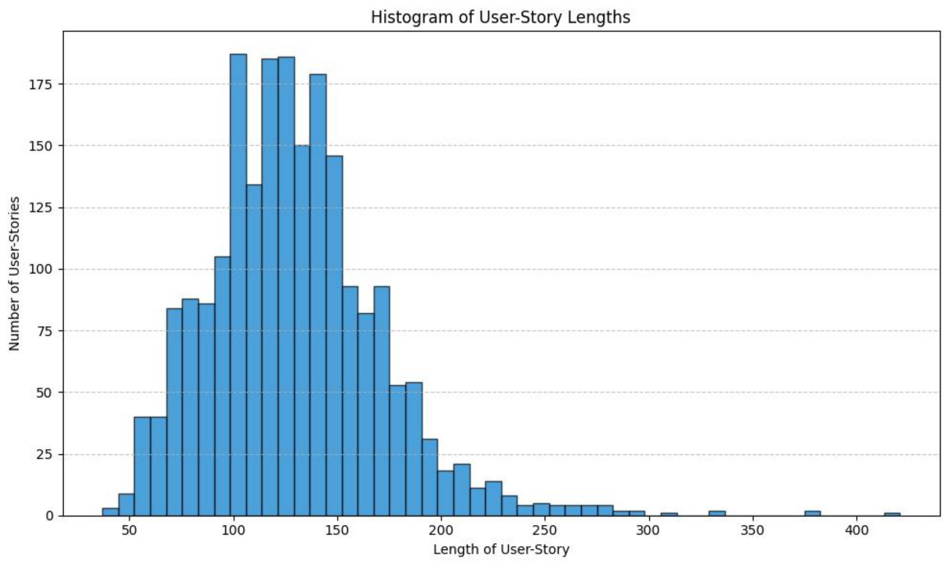

Fig. 3.2. User-Story Text Length Distribution

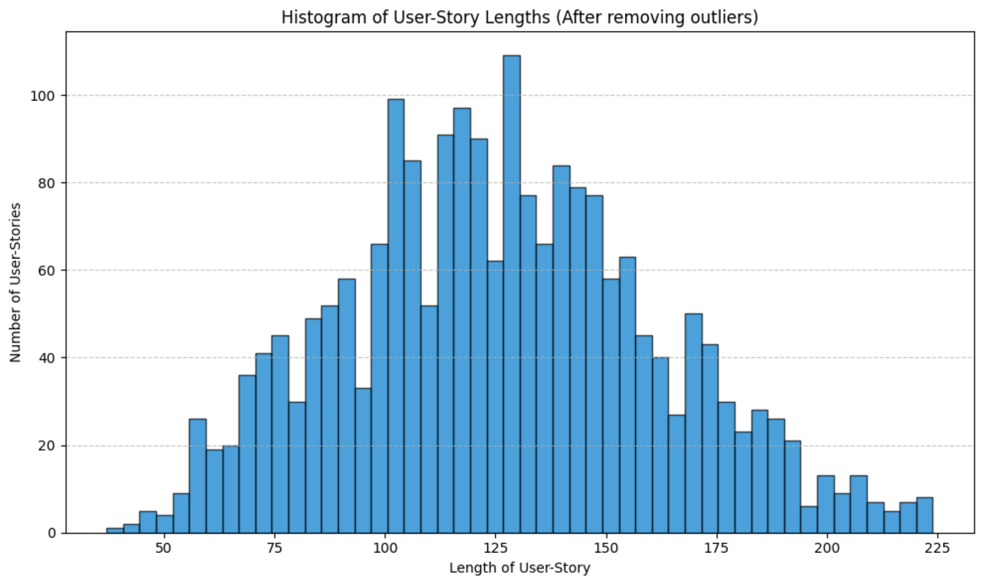

Fig. 3.3. User-Story Text Length Distribution (after removing outliers)

for generalisation to real-world scenarios, reducing the risk of overfitting and resource efficiency. Two approaches are used for the classification task and the one with the highest accuracy is selected. Both approaches use a self-training method for modelling the unseen data for classification using a subset of labelled datasets. We use GAN-BERT [48] based SSL and IBM-ZSL boosting [47] for classifying the unlabelled data rows and then comparing the performance. To apply these approaches the data needs to be divided into *test* and *unlabelled* sets. The algorithms use the *test* (labelled data) for training and *unlabelled* for self-training.

SSL-based approach: Setting up GAN-BERT requires fine-tuning the hyperparameters and slicing the labelled dataset into *test* and *labelled* sets. The *tagged* set is as tiny as 1% to 5% of the *test* set.

GAN-BERT uses a Discriminator-Generator setting for fine-tuning the training phase. For unlabelled data, we use *UNK* instead of *Unknown* as the class label as per the implementation. A summary of hyperparameters used has been summarised in Table 3.2. The average accuracy after training GAN=BERT over the test set is approximately 73%.

Parameter	**Value**
Transformer Parameters	
Max Sequence Length	256
Batch Size	32
GAN-BERT Specific Parameters	
Number of Hidden Layers (Generator)	1
Number of Hidden Layers (Discriminator)	1
Noise Size	100
Dropout Rate (Discriminator Input)	0.2
Apply Balance	False
Optimisation Parameters	
Learning Rate (Discriminator)	5×10^{-5}
Learning Rate (Generator)	5×10^{-5}
Epsilon	1×10^{-8}
Number of Training Epochs	20
Apply Scheduler	True
Warm-up Proportion	0.1
Adopted Transformer Model	
Model Name	**bert-base-uncased**

Table 3.3. Parameter values for GAN-BERT model-based data labelling

ZSL-based approach: Setting up zero-shot models with self-training requires the model file path or the name using the Hugging-Face's zero-shot natural language inference models. In addition to the *test* and *unlabelled* files the algorithm also requires a text file named *class_names* having all the class label names separated by a newline. Table 3.3 summarises all the parameters used for iterative self-training of the base models.

Parameter	**Value/Type**
dataset_directory	user_stories
zero-shot_model	**sileod/deberta-v3-large-tasksource-nli**
num_iterations	3
dataset_subset_size	10000
sample_ratio	0.005
negative_sampling_strategy	take_random
learning_rate	2e-5
train_batch_size	4
infer_batch_size	4
max_length	256
seed	42

Table 3.4. Parameter values for zero-shot model based with iterative self-training data labelling

The accuracy achieved is close to 93%, and the classification report for the same over the *test* set is summarised in Table 3.4. The *negative_sampling_strategy* used is *take_random* which gave better performance than *take_all* as the parameter value. It is concluded that the *Accessibility* class label has the lowest F1-score and *Chat* has the lowest *Recall* value. The stakeholders present in *Accessibility* are mainly customers and the feature development might relate to a different aspect for the SD team. Due to this the *Accessibility* label is often a point of discussion and it often shares characteristics from the other classes present in our dataset. The low *Recall* values are acceptable in the non-mission critical distribution of tasks among the team members.

Class	**Precision**	**Recall**	**F1-Score**	**Support**
Accessibility	0.77	0.91	0.83	11
Aggregation	0.92	1.00	0.96	11
Authentication	0.90	0.90	0.90	10
Chat	1.00	0.83	0.91	18
Consistency	1.00	1.00	1.00	11
Contacts	1.00	1.00	1.00	16
Overall Metrics				
Accuracy	0.94			
Macro Avg	0.93	0.94	**0.93**	77
Weighted Avg	0.94	0.94	0.94	77

Table 3.5. Classification report of the fine-tuned zero-shot model over the test dataset

Handling Class Imbalance: After the dataset has been labelled, the class distribution is found to be imbalanced as depicted in Table 3.6. For most algorithms to work properly, all classes in the data should be equally represented. When working with unequally represented classes, the model might struggle to accurately classify the data rows, which could lead to poor results for that class. However, when trained on the classification task, the TBLMs outperform other deep learning models upon imbalanced datasets. We perform class balancing (CB) only on the training dataset.

We implement two methods for dealing with imbalanced datasets. The first one is during the dataset split for training the model we use stratified sampling instead of random sampling to improve the effectiveness of the partitioned data samples [61]. Secondly, we determine the class weights for each label in the training set and then send those weights to the loss function. Table 3.6 summarises the class weights for all our six class labels.

The data distribution of the final dataset used for further training the TBLMs is depicted in Table 3.6 with their numeric label id values. Accuracy can fail to be the appropriate statistic when working with unbalanced datasets. Macro-averaged F1-score matrices are used for model evaluations [62].

3.4 Fine Tuning Pre-trained Model

To answer RQ-2 we use the pre-trained model available for text classification in the HG library. For a variety of activities, a pre-trained model has been trained on a dataset particular to the task at

Label Id	Label	Count	Weight
0	Chat	63	5.5303
1	Contacts	500	0.6952
2	Aggregation	604	0.5752
3	Accessibility	398	0.8752
4	Consistency	452	0.77
5	Authentication	70	4.9659

Table 3.6. Distribution of class labels on the final dataset after data labelling with the corresponding label-id and their respective weights

hand. This effective training method is called fine-tuning. By doing so, we use cutting-edge models by not having to spend time and resources training them from scratch, which also lowers computing costs. Following are the steps used for fine-tuning a pre-trained model.

Dataset setup: The data present in Pandas DF needs to be first partitioned into *train*, *validation* and *test* sets. Further, the label class names need to be converted to their corresponding numerical representation using the label IDs as mentioned in Table 3.6. The *Trainer* API from HG requires the dataset to be loaded into a dictionary format with all three partitioned data sets as the dictionary keys. We use 70% of the dataset for training the models and the remaining 30% is divided into test and validation sets. As the data classes are imbalanced we use *stratify* parameter of *train_test_split* method over the *label* column to sample evenly all the class labels in the dataset split.

Tokenization: Tokenizers transform unprocessed text into a form that machine learning algorithms can comprehend. The US data text present in the dataset is of variable length. We use the tokenizer to process the text data for truncation and padding in the variable US lengths. We use the same tokenizer as the pre-trained model to achieve consistency with the pre-training, models rely on the presence of the special tokens for processing and the sub-word tokenization. We use *fast tokenizers* from HG as they offer higher speed, efficiency, and flexibility than the traditional Python-based tokenizers. The output of the tokenizers is *input_ids* and *attention_mask*. The *input_ids* are the integer IDs based on the predefined vocabulary of the model. The *attention_mask* is a kind of binary mask that specifies which tokens the model should focus on. For each genuine token, it is normally set to 1, and for padding tokens, to 0. Both the output from the tokenizer is stored in the dataset dictionary.

Hyperparameter Tuning: Due to their prior training, the pre-trained models offer a robust foundation. However, effective hyperparameter tuning is required to adapt these models to particular workloads. It makes sure that the model completes the new task as efficiently as possible, generalises well to unexplored data, and makes effective use of computational resources. Overfitting is more likely to happen while fine-tuning over a smaller dataset. This risk can be reduced and the model can be kept from simply memorising the training data by changing hyperparameters like the learning rate, dropout rate, and batch size. We use a randomised search over the hyperparameter space to find the optimal values.

Model Training: To learn features from our US dataset, the model must be trained, which involves changing its internal parameters. The model parameters are updated based on the difference between the prediction and actual values of the class labels which is the error. This error is iteratively minimised using an optimisation algorithm called gradient descent [63]. The HG library provides a *Trainer* class for training the pre-trained models on our custom dataset. We start by loading the model with its appropriate HG name identifier and specifying the number of labels which is six in our case. The training arguments contain the hyperparameter configurations such as learning rate, number of epochs, batch size and weight decay.

The *Trainer* class takes the training argument object with the model, dataset, tokenizer and metrics computation function. If the model has stopped learning when the prediction error stays the same after a certain time, we stop our training as the model has converged. To ensure model robustness, we apply a cross-validation technique. According to this strategy, the dataset is split into 'k' equal-sized subgroups. The other 'k-1' subsets act as training data, and one subset is utilised for validation. Each subgroup is utilised for validation once through 'k' iterations of this process. A comprehensive performance measure is produced by averaging the outcomes from these "k" repetitions.

3.5 Model Evaluation Methods

The comprehensive assessment of pre-trained models is a key component of the methodology used for this study. This guarantees that the models created are reliable, resilient, and generalised in addition to being precise. The evaluation strategies employed for answering RQ-2 are described below in more detail:

3.5.1 Performance Metrics

We use several measures that give us a complete picture of the performance of our models to assess their efficacy. Taking imbalanced class labels into consideration we use macro-averaged matrices so that all classes equally contribute to the final averaged metric scores [62]. The formula for calculating the matrices used is described below:

True positive (TP) refers to the instances that are correctly classified into their actual category.

True negative (TN) for a particular category, let's say "Authentication," denotes that a user story that does not fall under "Authentication" is classed under another category.

False positive (FP) refers to the instance when the US is incorrectly classified into a category it doesn't belong to.

False negative (FN) refers to the instance when the US which belongs to a certain category is incorrectly classified into a different category.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{3.1}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{3.2}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{3.3}$$

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3.4}$$

$$\text{Macro F1-Score} = \frac{1}{N} \sum_{i=1}^{N} \text{F1-Score}_i \tag{3.5}$$

Where *i* is the F1-Score for the *i-th* class.

3.5.2 Comparative Analysis

We train multiple pre-trained models on the same dataset so that their performance can be directly compared. This method makes it easier to determine which algorithms are best for the task at hand. The comparison takes into account several parameters, including accuracy, F1-score, and computing efficiency which directly relate to the strengths and weaknesses of the models. This comprehensive study ensures that the ideal model is selected for a particular setting, as well as its interpretability for users/stakeholders and flexibility to accommodate changing data distributions.

Benchmarking Models: It is insufficient to simply compare models based on how well they perform on a single job in light of ever-larger language models. The computational cost associated with a particular model should be taken into account. The computational costs for training a model or using it in inference often depend only on the necessary memory and the necessary time for a certain computing environment (such as a particular type of GPU). Therefore, it is crucial for our research that we can accurately test language models in terms of both performance and required memory. We use the *PyTorchBenchmark* class from the HG library for benchmarking the models compared in our study. We use the batch size hyperparameter for benchmarking our models.

The memory consumed by a language model is dependent upon the maximum sequence length and the batch size parameters. We have computed a constant value for the maximum length pa-

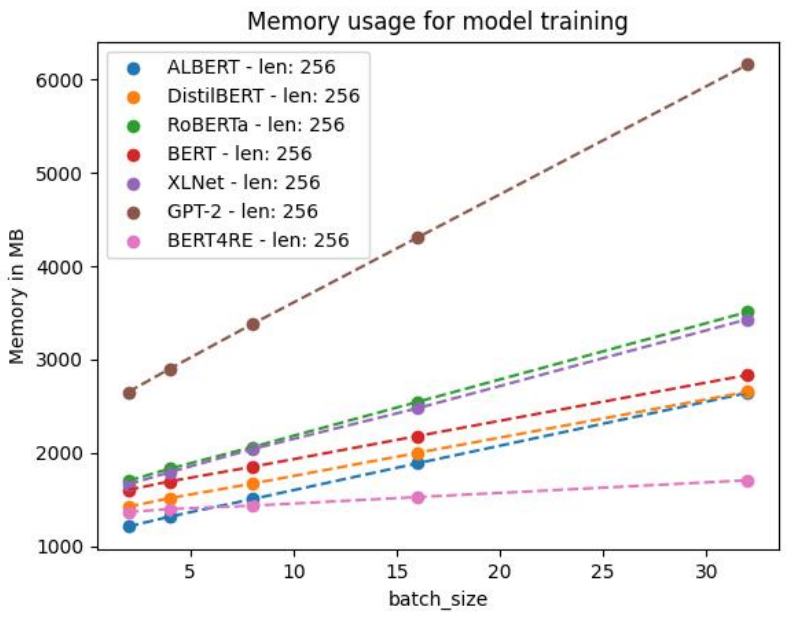

Fig. 3.4. Memory usage benchmarking of all the models used in our study.

rameter of all the TBLMs. We benchmark the models based on the varying batch sizes. Figure 3.4 illustrates the memory consumption based on different batch sizes.

The second benchmark is executed on the speed of the model and is measured in seconds. We use the batch size as 16 as per the plot in Fig 3.4. We use the plot to identify the best model for the predefined token size of 256. We use the benchmarking plot to select the best model for classification in the next chapter. Fig 3.5 depicts the time usage based on different token sizes/maximum lengths keeping the batch size constant.

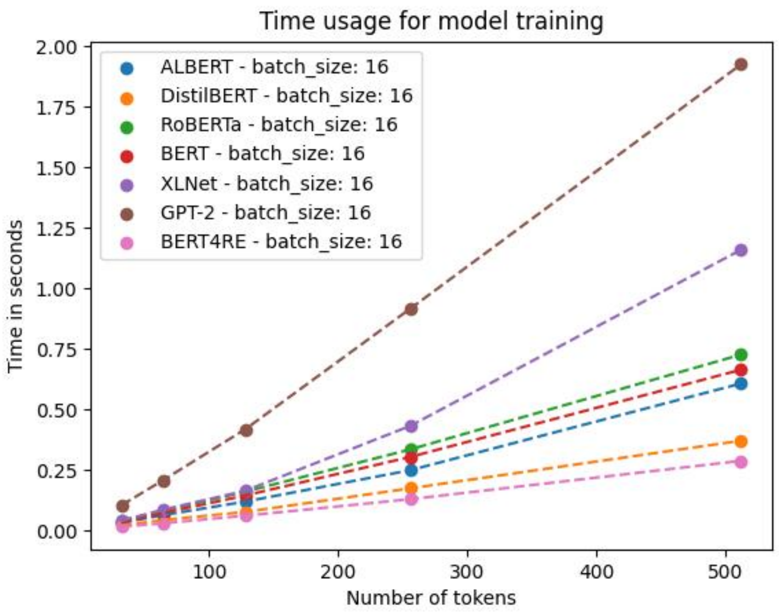

Fig. 3.5. Time usage benchmarking of all the models used in our study keeping the batch size constant.

Chapter 4

Results Analysis

This chapter contains the findings from the analysis of several cutting-edge models, including ALBERT, DistilBERT, RoBERTa, XLNet, GPT-2, and BERT4Re. Accuracy, precision, recall, and macro-averaged F1 scores are some of the evaluation criteria employed in this investigation. We also investigate these models' computational properties, looking at their total number of parameters, training time, and inference time. This comprehensive evaluation tries to shed light on each model's effectiveness and performance within the framework of our particular goal. We first analyse the individual model's performance and then do a comparative study using the output of all the fine-tuned models aggregated in Table 4.1. The *Precision*, *Recall* and *F1-Score* metrics detailed are macro-averaged [62] and the training runtime is in seconds. To prevent the model from overfitting over the training dataset *EarlyStopping* [64], the callback method is used with a value of three which stops the training process after the model fails to optimize further.

ALBERT which is a simplified form of BERT, contains fewer parameters, which reduces the time needed for training and inference. In contrast to its larger rivals, the parameter reduction results in a slight performance metric compromise. We validate the same as the overall accuracy of this model among the other BERT-based models is the lowest. The fine-tuning training runtime is the second largest in our analysis.

DistilBERT which is a distilled version of BERT, keeps the majority of BERT's capabilities while being quicker and requiring fewer parameters. We found it to have a balance between effectiveness and performance. All the macro-averaged matrices' score is close to the best model. This model trains very quickly but fails to achieve a better accuracy score.

RoBERTa often beats BERT in accuracy and F1-score matrices because it is constructed using BERT's architecture but using additional data and training. However, given its rigorous training schedule, this model requires more time to train. It achieves best results for *Accuracy* and *Precision* scores. The number of epochs for training the model is the lowest among all the models used in our analysis. Due to the increase in parameters from the DistilBERT model, it takes more training runtime to converge. We found this model to be best for our use case in terms of time and space complexity as per the benchmarking plots constructed in the previous chapter.

BERT base model is the foundation of all the above-mentioned models. We used this model in our study to test against its lighter alternatives. We found this model to be best in terms of *Recall* and *F1-score* matrices. To achieve this the model takes double the number of training epochs than the RoBERTa model. The training runtime is hence doubled then the best model as compared in our experiments.

XLNet model frequently performs better when using a permutation-based training technique, especially for tasks requiring comprehension of the bidirectional context. Compared to models like DistilBERT, its overall number of parameters and training time are found to be larger. The increase in the number of parameters makes the model training slower. The results obtained using this model fall short on all the performance as well as computational matrices. This confirms the fact that bigger models need more resources and a huge amount of data to perform better which in our study was not present.

GPT-2 model's main purpose is text generation and has different advantages and disadvantages than the BERT-based models. We fine-tuned the GPT-2 model on our dataset to study if the decoder-stacked models performed better than the encoder-stacked models in language inference text classification. On the same grounds as the XLNet model, this model also has a huge number of parameters and it only supports left context as it predicts the next word in the context instead of the BERT-based model which learns contextual representations.

BERT4Re is a variant of BERT retrained on requirements engineering data. This is a domain-specific model for the task at hand. We use this model to answer RQ-4 which evaluates whether the domain-specific or the generic types of the TBLMs performs best. The performance metrics fall short of all the other BERT-based models as the US are written by varied type of stakeholders/user using different writing style and jargon. Considering this randomness and the retrained requirements data used for this model it is evident that the domain-specific models do not outperform the general-purpose models.

In summary, after comparing the models in terms of performance and computational matrices we found that models with fewer parameters and quicker training timeframes, like DistilBERT and ALBERT, excel in terms of computing efficiency. RoBERTa and XLNet models need more processing resources and are best suited when performance is prioritised. The US classification is done at the time of sprint planning using the agile SD methodology. The time complexity of the model takes priority rather than the precision in the non-mission critical US classification often between the teams or individual software developers. Aggregating these factors the RoBERTa model is recommended for the US classification task on the given dataset.

Model Name	Model Training				Model Evaluation			
	Run-time	Epochs	Learn-ing Rate	Batch Size	Accu-racy	Preci-sion	Recall	F1-Score
ALBERT	591.02	16	2.19e-05	16	0.7371	0.7876	0.6649	0.7051
Distil-BERT	**473.73**	13	4.71e-05	32	0.7482	0.7827	0.7509	0.7481
RoBERTa	578.64	**11**	4.89e-05	16	**0.7998**	**0.8013**	0.7533	0.7501
BERT	971.01	18	2e-05	8	0.7666	0.7359	**0.7618**	**0.7627**
XLNet	1187.75	22	1.3e-5	4	0.7409	0.7043	0.6791	0.6879
GPT-2	2119.92	17	2e-5	2	0.6997	0.6672	0.6305	0.6427
BERT4RE	703.60	21	3.97e-5	16	0.6736	0.7135	0.6279	0.6587

Table 4.1. An illustration of model training and evaluation parameter/metric values of the fine-tuned TBLMs.

Chapter 5

Evaluation and Discussion

In this chapter, we address the research questions based on the comprehensive analysis of different pre-trained TBLMs. The strategies and techniques used in the creation of this study were carefully chosen. It is crucial to recognise that the research prioritised the use of TBLMs over machine or deep learning techniques despite having access to a smaller dataset. This focus was impacted by the insightful advice and recommendations shared by the supervisor who understood the efficacy of large language models to solve the study objectives. The industry demands TBLMs in natural language processing jobs and ongoing technological improvements are reflected in this choice.

The related work introduced the concept of validating the US as per their structure and INVEST principles but did not directly classify them into logical categories which can be beneficial for efficient SD processes using agile. The major constraints of the related work lie in the implemented word embedding and the language inference. The models fine-tuned in our study have bidirectional and unidirectional contextual embeddings, giving an edge over the models to classify the US dataset. With an awareness of these drawbacks, this study seeks to provide insight into the advantages and disadvantages of TBLMs in the particular setting of requirement labelling and classification.

5.1 Answers to the research questions

RQ-1: What is the level of effectiveness of semi-supervised and zero-shot learning using self-training for labelling the given user-story dataset?

The available labelled dataset for our study is limited. Therefore, we tried both the semi-supervised and zero-shot learning-based approaches for the textual dataset labelling task. The performance matrices for both operations have a huge difference of about 25% in F1-score values. This gap suggests that the semi-supervised based GAN-BERT algorithm is dependent on the feature extracted from the labelled *test* set and does not have language inference capability in contrast to the zero-shot model. The ZSL approach not only learns the feature from the *test* set but also applies previously trained natural language inference mechanisms to accurately identify and classify unseen data samples. Thus, for a text dataset labelling ZSL based approach using the self-training capability is effective and performs best in our evaluation.

RQ-2: How do various fine-tuned transformer-based language models compare in terms of their performance for classifying the given user-story dataset?

The model used for fine-tuning the given user-story dataset is selected from both the classes of the transformers-based model. We also use a domain-specific model which is retrained on requirements engineering datasets. We perform a training process followed by evaluation for generating the performance metrics. We found that the RoBERTa model performs the best in terms of the performance metrics scores. The efficiency of the model helps in integrating it directly into the Agile SD workflow. Autoregressive models like XLNet and GPT-2 do not perform well in classifying the US due to their design for language generation tasks mainly generating longer coherent texts. The BERT-based model is designed for fine-tuning and hence delivers superior performance. We also conclude that the size of the model does not guarantee performance advantages if the amount of training dataset and resources are limited.

RQ-3: How does the choice of the evaluation metrics impact the understanding and assessment of transformer-based language model performance in classifying the given user-story dataset?

TBLMs are trained on a huge corpus of data. Different performance matrices focus on particular aspects of a model evaluation. In the process of multi-class classification, these metrics play an important role in understanding the analytics related to each class as well as the combination of them which often is not possible by just relying on an accuracy score. The model's application highly depends on the performance matrix into consideration [62]. If the model is used in a mission-critical environment where a high *Recall* score is important then the model selection is dependent on that particular score. In our case we want to have the optimal classification and the misallocation can be overlooked. Therefore, we consider the macro-averaged *F1-score* results for model selection. On the other hand, if the class label distribution in the dataset is uniform then the micro-averaged or weighted-average scores are more relevant.

RQ-4: How does the domain-specific model perform in comparison with the generic model?

The generic models are trained on a vast amount of data whereas the domain-specific models are trained on a portion of data that is relevant to that domain. Due to these the domain-specific models lack the generalizability which is required when training on a new dataset. In our analysis, it is found that the generic models outperform the domain-specific ones. As different stakeholders write the US it needs to be analysed without weight biases present in the domain-specific models.

5.2 Comparing related approaches and limitations of the project

For a comparison with the related work as discussed in Chapter 2, our study has major upgrades. Our project extends the model developed by Kochbati [49] which uses term frequency and inverse term frequency-based embeddings. These embeddings are easy to compute but do not capture the syntactic and semantic meaning of words in the document. Instead of word-based embedding, we use contextualized word representations that work over a sentence or document level.

To the best of our knowledge, we did not find any related work for user-story classification using the TBLMs. Our study is a new contribution to this research field. Even though many studies have used user-story for classification they mainly depend on a labelled dataset and address the structural or quality issues [29] [43] [44]. In our study, we develop a data annotation mechanism that is capable of labelling the given user-story dataset with little or no labelled data and manage to achieve optimal accuracy.

The implementation of different performance matrices is in line with the related studies of requirements classification [40]. We extend the usage of macro-averaged scores instead of accuracy for better refinement and analysis of the model's performance. The generic models are found to be more efficient and robust for the US classification on a given dataset. Thereby the results obtained comply with the related work which evaluates domain-specific and generic models [45].

The use of TBLMs for the classification of the US is a potent tool for agile SD. The categorisation increases efficiency and limits the turnaround time of a component or feature implementation in the SD. Due to the use of domain-specific jargon and writing style, the US contains dissimilar contextual word embedding which is harder to label into a certain class. The primary limitation lies in the reliance on the high quantity and quality of the dataset. The TBLMs used in the study require high volumes of data to train upon.

The findings also revealed that performance varied between models. GPT-2 underperformed BERT-based models in the classification task. This emphasises the model variability of choosing a model based on the particulars and specifications of the task at hand. Due to their intricacy, the TBLM frequently function as *black-boxes*. It might be quite important for applications that demand explainability to be able to understand why a given model generated a particular prediction which is an Interpretability limitation of our study.

Threats to Validity lists potential threats that could compromise the validity of the project findings and describes the measures taken to address them. The threats to validity can be of two types for instance internal and external. The internal validity describes how well the study was performed and the quality of it [65]. To mitigate internal threats to validity we use the stakeholder information and background knowledge for validating the labelled US dataset. This is done to mitigate any error propagation using the self-training-based data labelling models. The collected US are then preprocessed for removing contents that do not add up in the model training such as punctuations, stop words, numbers and dates. We use a common *max-length* parameter as an input to the TBLM for all the US present in the dataset after removing outliers. The interpretation and debugging of the TBLM are done using the training and validation loss values.

The focus of our study primarily used base models instead of their larger version due to resource constraints. While the approach yields good accuracy it is crucial to recognise that the use of other TBLM or retraining a custom one may result in even higher performance. The results or findings of the study might not apply to other TBLMs outside of those that were specifically studied in the study due to the aforementioned restriction on internal validity. Future studies should compare the effectiveness of a wider variety of language models while using various word embeddings to increase the internal validity of their findings. The limits in internal validity found in this study may be addressed by using this approach, which could offer a more thorough and rigorous grasp of the research topic.

According to the context in which the study was conducted, external validity measures how generalizable the study is. The US collected from both the data sources are written by experts and stakeholders. These US follow the INVEST principles and are of high quality. Therefore the data collected is in line with the final operating procedure of the TBLMs for classification task. The models trained on the industrial dataset as encountered in software engineering companies. These steps make the study robust and generalised for application in related industries. By acknowledging this wider application and relevance, it suggests that the study's findings can be applied to other analogous industry practices and datasets that are frequently encountered in real-world contexts.

Chapter 6

Conclusion

6.1 Summary of Achievements

In the course of this study, we set out to investigate the potential of transformer-based models for user story classification. These are this dissertation's main accomplishments:

- Models based on BERT and GPT-2 architecture were thoroughly examined. This gave a fundamental grasp of their design, operation, and applicability in the context of NLP. We also found that the RoBERTa model performed best than other BERT and GPT models in user story classification both in terms of performance and resource utilisation.
- A methodological advancement is achieved in optimising these models specifically for the classification of the given US dataset. This research not only demonstrated the flexibility of transformer-based models but also their potential for tasks that are specialised in the software engineering domain.
- Using an empirical evaluation various TBLMs used in our study resulted in highlighting the strengths and limitations of each model offering valuable insights for future studies or enhancements.
- An end-to-end workflow for the classification of a given unlabeled US dataset is implemented using TBLMs which can be seamlessly applied in real-world SD processes.

6.2 Future Work

Although our study has advanced our understanding of the use of TBLMs for the US classification, there are still several directions worth exploring in the future. The following recommendations for further research can be made based on the constraints and threats to validity mentioned in the previous chapter.

- Future research should go further into understanding the underlying causes for such discrepancies given the observed performance variations among models like GPT-2 and BERT-based models. As a result, recommendations or frameworks for choosing a model based on requirements may be implemented. The TBLMs present difficulties in applications requiring interpretability due to their ”black-box” characteristics. Future research should concentrate on creating methods or instruments that improve the interpretability of these models.
- Due to resource limitations, *base* (smaller version) models were predominantly used in the investigation. Improved performance might result from investigating more complex iterations of these models or perhaps altogether new architectures. As a result, the capabilities of different TBLMs in the US classification would be more thoroughly understood.
- TO address the threats to validity future studies should consider comparing a wider range of models, experiments with various word embeddings and expanding the dataset to include the US from diverse sources which would test the robustness of the models in varied scenarios. Future research should examine optimisation strategies that enable the use of more complex models without considerably raising computational costs in light of the resource constraints encountered in our study.

In conclusion, this study has proven to be both difficult and gratifying to label a given US dataset and evaluate different TBLMs for the US classification task. The results obtained from this study open the way for more effective and efficient software development procedures, and there is tremendous room for growth in this field in the years to come.

References

[1] G. Lucassen, F. Dalpiaz, J. M. E. Van Der Werf, and S. Brinkkemper, “Forging high-quality user stories: Towards a discipline for agile requirements,” in *2015 IEEE 23rd international requirements engineering conference (RE)*, IEEE, 2015, pp. 126–135.

[2] M. R. Resketi, H. Motameni, H. Nematzadeh, and E. Akbari, “Automatic summarising of user stories in order to be reused in future similar projects,” *IET Software*, vol. 14, no. 6, pp. 711–723, 2020.

[3] Y. Wautelet, S. Heng, M. Kolp, and I. Mirbel, “Unifying and extending user story models,” in *Advanced Information Systems Engineering: 26th International Conference, CAiSE 2014, Thessaloniki, Greece, June 16-20, 2014. Proceedings 26*, Springer, 2014, pp. 211–225.

[4] G. Lucassen, F. Dalpiaz, J. M. E. van der Werf, and S. Brinkkemper, “Improving agile requirements: The quality user story framework and tool,” *Requirements engineering*, vol. 21, pp. 383–403, 2016.

[5] D. Khurana, A. Koli, K. Khatter, and S. Singh, “Natural language processing: State of the art, current trends and challenges,” *Multimedia tools and applications*, vol. 82, no. 3, pp. 3713–3744, 2023.

[6] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding,” *arXiv preprint arXiv:1810.04805*, 2018.

[7] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, *et al.*, "Improving language understanding by generative pre-training," 2018.

[8] Y. Liu, M. Ott, N. Goyal, *et al.*, "Roberta: A robustly optimized bert pretraining approach (2019)," *arXiv preprint arXiv:1907.11692*, vol. 364, 1907.

[9] P. Abrahamsson, O. Salo, J. Ronkainen, and J. Warsta, "Agile software development methods: Review and analysis," *arXiv preprint arXiv:1709.08439*, 2017.

[10] S. Al-Saqqa, S. Sawalha, and H. AbdelNabi, "Agile software development: Methodologies and trends.," *International Journal of Interactive Mobile Technologies*, vol. 14, no. 11, 2020.

[11] J. Jia, X. Yang, R. Zhang, and X. Liu, "Understanding software developers' cognition in agile requirements engineering," *Science of Computer Programming*, vol. 178, pp. 1–19, 2019.

[12] G. Lucassen, F. Dalpiaz, J. M. E. v. d. Werf, and S. Brinkkemper, "The use and effectiveness of user stories in practice," in *Requirements Engineering: Foundation for Software Quality: 22nd International Working Conference, REFSQ 2016, Gothenburg, Sweden, March 14-17, 2016, Proceedings 22*, Springer, 2016, pp. 205–222.

[13] B. Wake, "Invest in good stories, and smart tasks," *Retrieved December*, vol. 13, p. 2011, 2003.

[14] I. K. Raharjana, D. Siahaan, and C. Fatichah, "User stories and natural language processing: A systematic literature review," *IEEE access*, vol. 9, pp. 53 811–53 826, 2021.

[15] G. Gupta and S. Malhotra, "Text document tokenization for word frequency count using rapid miner (taking resume as an example)," *Int. J. Comput. Appl*, vol. 975, p. 8887, 2015.

[16] D. Khyani, B. Siddhartha, N. Niveditha, and B. Divya, “An interpretation of lemmatization and stemming in natural language processing,” *Journal of University of Shanghai for Science and Technology*, vol. 22, no. 10, pp. 350–357, 2021.

[17] D. Kumawat and V. Jain, “Pos tagging approaches: A comparison,” *International Journal of Computer Applications*, vol. 118, no. 6, 2015.

[18] K. Kowsari, K. Jafari Meimandi, M. Heidarysafa, S. Mendu, L. Barnes, and D. Brown, “Text classification algorithms: A survey,” *Information*, vol. 10, no. 4, p. 150, 2019.

[19] M. J. Er, R. Venkatesan, and N. Wang, “An online universal classifier for binary, multi-class and multi-label classification,” in *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, IEEE, 2016, pp. 003 701–003 706.

[20] M. I. Jordan and T. M. Mitchell, “Machine learning: Trends, perspectives, and prospects,” *Science*, vol. 349, no. 6245, pp. 255–260, 2015.

[21] V. Nasteski, “An overview of the supervised machine learning methods,” *Horizons. b*, vol. 4, pp. 51–62, 2017.

[22] N. Rahimi, F. Eassa, and L. Elrefaei, “An ensemble machine learning technique for functional requirement classification,” *symmetry*, vol. 12, no. 10, p. 1601, 2020.

[23] J. E. Van Engelen and H. H. Hoos, “A survey on semi-supervised learning,” *Machine learning*, vol. 109, no. 2, pp. 373–440, 2020.

[24] O. T. Nartey, G. Yang, J. Wu, and S. K. Asare, “Semi-supervised learning for fine-grained classification with self-training,” *IEEE Access*, vol. 8, pp. 2109–2121, 2019.

[25] A. Blum and T. Mitchell, “Combining labeled and unlabeled data with co-training,” in *Proceedings of the eleventh annual conference on Computational learning theory*, 1998, pp. 92–100.

[26] K. Berahmand, S. Haghani, M. Rostami, and Y. Li, “A new attributed graph clustering by using label propagation in complex networks,” *Journal of King Saud University-Computer and Information Sciences*, vol. 34, no. 5, pp. 1869–1883, 2022.

[27] M. Alloghani, D. Al-Jumeily, J. Mustafina, A. Hussain, and A. J. Aljaaf, “A systematic review on supervised and unsupervised machine learning algorithms for data science,” *Supervised and unsupervised learning for data science*, pp. 3–21, 2020.

[28] T. Hofmann, “Unsupervised learning by probabilistic latent semantic analysis,” *Machine learning*, vol. 42, pp. 177–196, 2001.

[29] B. Kumar, U. K. Tiwari, D. C. Dobhal, and H. S. Negi, “User story clustering using k-means algorithm in agile requirement engineering,” in *2022 International Conference on Computational Intelligence and Sustainable Engineering Solutions (CISES)*, IEEE, 2022, pp. 1–5.

[30] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, J. Pineau, *et al.*, “An introduction to deep reinforcement learning,” *Foundations and Trends® in Machine Learning*, vol. 11, no. 3-4, pp. 219–354, 2018.

[31] H. Sultanov and J. H. Hayes, “Application of reinforcement learning to requirements engineering: Requirements tracing,” in *2013 21st IEEE International Requirements Engineering Conference (RE)*, IEEE, 2013, pp. 52–61.

[32] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[33] K. Cho, B. Van Merriënboer, C. Gulcehre, *et al.*, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[34] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[35] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: Smaller, faster, cheaper and lighter," *arXiv preprint arXiv:1910.01108*, 2019.

[36] Y. Liu, M. Ott, N. Goyal, *et al.*, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

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

[38] P. He, X. Liu, J. Gao, and W. Chen, "Deberta: Decoding-enhanced bert with disentangled attention," *arXiv preprint arXiv:2006.03654*, 2020.

[39] S. Black, S. Biderman, E. Hallahan, *et al.*, "Gpt-neox-20b: An open-source autoregressive language model," *arXiv preprint arXiv:2204.06745*, 2022.

[40] E. Dias Canedo and B. Cordeiro Mendes, "Software requirements classification using machine learning algorithms," *Entropy*, vol. 22, no. 9, p. 1057, 2020.

[41] A. Likas, N. Vlassis, and J. J. Verbeek, "The global k-means clustering algorithm," *Pattern recognition*, vol. 36, no. 2, pp. 451–461, 2003.

[42] S. Qaiser and R. Ali, "Text mining: Use of tf-idf to examine the relevance of words to documents," *International Journal of Computer Applications*, vol. 181, no. 1, pp. 25–29, 2018.

[43] M. Jurisch, S. Böhm, and T. James-Schulz, “Applying machine learning for automatic user story categorization in mobile enterprises application development,” 2020.

[44] Z. Kurtanović and W. Maalej, “Automatically classifying functional and non-functional requirements using supervised machine learning,” in *2017 IEEE 25th International Requirements Engineering Conference (RE)*, Ieee, 2017, pp. 490–495.

[45] W. Alhoshan, A. Ferrari, and L. Zhao, “Zero-shot learning for requirements classification: An exploratory study,” *Information and Software Technology*, vol. 159, p. 107 202, 2023.

[46] S. M. Jain, “Hugging face,” in *Introduction to Transformers for NLP: With the Hugging Face Library and Models to Solve Problems*, Springer, 2022, pp. 51–67.

[47] A. Gera, A. Halfon, E. Shnarch, Y. Perlitz, L. Ein-Dor, and N. Slonim, “Zero-shot text classification with self-training,” in *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, Dec. 2022, pp. 1107–1119. [Online]. Available: `https://aclanthology.org/2022.emnlp-main.73`.

[48] D. Croce, G. Castellucci, and R. Basili, “Gan-bert: Generative adversarial learning for robust text classification with a bunch of labeled examples,” 2020.

[49] T. Kochbati, S. Li, S. Gérard, and C. Mraidha, “From user stories to models: A machine learning empowered automation.,” *MODELSWARD*, vol. 10, p. 0 010 197 800 280 040, 2021.

[50] F. Dalpiaz, I. Van Der Schalk, S. Brinkkemper, F. B. Aydemir, and G. Lucassen, “Detecting terminological ambiguity in user stories: Tool and experimentation,” *Information and Software Technology*, vol. 110, pp. 3–16, 2019.

[51] M. Galster, F. Gilson, and F. Georis, “What quality attributes can we find in product backlogs? a machine learning perspective,” in *Software Architecture: 13th European Conference, ECSA 2019, Paris, France, September 9–13, 2019, Proceedings 13*, Springer, 2019, pp. 88–96.

[52] R. Barbosa, D. Januario, A. E. Silva, R. Moraes, and P. Martins, “An approach to clustering and sequencing of textual requirements,” in *2015 IEEE International Conference on Dependable Systems and Networks Workshops*, IEEE, 2015, pp. 39–44.

[53] G. Lucassen, M. Robeer, F. Dalpiaz, J. M. E. Van Der Werf, and S. Brinkkemper, “Extracting conceptual models from user stories with visual narrator,” *Requirements Engineering*, vol. 22, pp. 339–358, 2017.

[54] Y. Wautelet, S. Heng, M. Kolp, I. Mirbel, and S. Poelmans, “Building a rationale diagram for evaluating user story sets,” in *2016 IEEE Tenth International Conference on Research Challenges in Information Science (RCIS)*, IEEE, 2016, pp. 1–12.

[55] S. P. Contributors, *User stories*, `https://github.com/solid/user-stories`, 2023.

[56] F. Dalpiaz, *Requirements data sets (user stories)*, version V1, Mendeley Data, 2018. DOI: `10.17632/7zbk8zsd8y.1`. [Online]. Available: `http://dx.doi.org/10.17632/7zbk8zsd8y.1`.

[57] W. McKinney *et al.*, “Data structures for statistical computing in python,” in *Proceedings of the 9th Python in Science Conference*, Austin, TX, vol. 445, 2010, pp. 51–56.

[58] 11166812, *Uom dissertation*, `https://github.com/balendra-singh/UoM_Dissertation`, 2023.

[59] H. Shelar, G. Kaur, N. Heda, and P. Agrawal, "Named entity recognition approaches and their comparison for custom ner model," *Science & Technology Libraries*, vol. 39, no. 3, pp. 324–337, 2020.

[60] F. J. P. Veitía, L. Roldán, and M. Vegetti, "User stories identification in software's issues records using natural language processing," in *2020 IEEE Congreso Bienal de Argentina (ARGENCON)*, IEEE, 2020, pp. 1–7.

[61] V. L. Parsons, "Stratified sampling," *Wiley StatsRef: Statistics Reference Online*, pp. 1–11, 2014.

[62] M. Grandini, E. Bagli, and G. Visani, "Metrics for multi-class classification: An overview," *arXiv preprint arXiv:2008.05756*, 2020.

[63] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[64] X. Ying, "An overview of overfitting and its solutions," in *Journal of physics: Conference series*, IOP Publishing, vol. 1168, 2019, p. 022 022.

[65] A. Ampatzoglou, S. Bibi, P. Avgeriou, M. Verbeek, and A. Chatzigeorgiou, "Identifying, categorizing and mitigating threats to validity in software engineering secondary studies," *Information and Software Technology*, vol. 106, pp. 201–230, 2019.