MANCHESTER 1824
The University of Manchester

Classification of app reviews for requirements engineering using deep learning models

A dissertation submitted to The University of Manchester for the degree of
Bachelor of Science in Computer Science
in the Faculty of Science and Engineering

Year of submission
2024

Supervisor: Dr. Liping Zhao
Author: Alp Erdal Yazici
Student ID: 10653872

School of Engineering

Contents

Word count: 12698

List of Figures and Tables

Abstract

With nearly 9 million applications available worldwide, platforms like Google Play Store and App Store have become essential for billions of people's lives. In the rapidly evolving digital ecosystem, app reviews serve as a crucial feedback mechanism. These reviews are a valuable resource for app vendors and developers, as they can provide insight into bugs, suggest new features, and offer feedback on existing ones. This report addresses the critical need for efficient and accurate classification of app reviews by implementing and comparing one deep learning model Convolutional Neural Networks (CNN), and two large language models DistilBERT and XLNet. The models classify app reviews into four unique categories: bug reports, feature requests, user experiences, and ratings. A series of experiments will be conducted to compare the performance of various models and examine the effects of preprocessing techniques, hyperparameter selection, dataset balance, and model complexity. Two crucial experiments were conducted to train and evaluate the models. The first one involved a train-test split of 75%-25% on the dataset, and the second one involved performing K-Flod Cross Validation on the dataset. The advanced language model XLNet had the best performance for the 75%-25% split with an average accuracy of 0.844 while the advanced language model DistilBERT had the worst performance with a score of 0.83. However, DistilBERT outperformed XLNet for the K-Fold Cross Validation, achieving an average accuracy score of 0.925, while XLNet exhibited the poorest performance with a score of 0.90. These results highlight the strengths of advanced language models in classifying app reviews and set the stage for further studies aimed at refining these models for practical use. This could significantly improve how user feedback is utilized in app development processes.

1 Introduction

1.1 Context and motivation

In the era of digital transformation, applications have become universal, significantly impacting our daily lives and routines. With nearly 9 million applications available on platforms like the Google Play Store and the App Store, the competition among applications has increased, making user feedback through app reviews more important than ever. These reviews are not just comments; they are rich, untapped veins of data that can provide critical insights into user experiences, preferences, and pain points. The reviews provided by users serve as a valuable resource for developers to plan feature maintenance activities [1]. Essentially, user reviews have become crucial in determining whether an application succeeds or fails. Developers rely heavily on this feedback to optimize functionality and enhance user satisfaction, which can significantly influence an app's success. Despite its importance, the task of manually processing this vast amount of feedback is daunting and inefficient as the volume of data continues to grow [2]. This makes the development of an automated, accurate classification system for app reviews essential. Such a system would allow developers to quickly identify and prioritize issues and updates, making the process more responsive to user needs and competitive pressures [3]. Additionally, an automated system would drastically reduce the time and resources spent on feedback analysis, enabling a more agile development cycle and fostering a deeper understanding of consumer behavior and preferences.

This study aims to implement and compare three different deep learning/large language models (CNN, DistilBERT and XLNet), to determine which model is the most useful for the developers in need. The models are assigned to classify the user reviews into four unique classes which are bug report, feature request, user experience and rating. Bug report represents descriptions of issues with the application that require correction, such as crashes, erroneous behavior, or performance problems. Feature request refers to user demands for additional functionality or content that is currently absent, for instance, in catalogues and games. User experience captures the subjective impressions and holistic experiences of users while engaging with the app, offering insights into how well the app meets their expectations and needs [4]. Lastly, ratings can be described as textual reflections of the numerical star rating system. However, it should be noted that ratings provide limited information as they mostly consist of expressions of approval, disapproval, negative criticism, or discouragement [5].

To analyze the performance of the classification models, evaluation metrics such as accuracy, precision, recall, and F1-score will be implemented. Accuracy represents the proportion of true results (both true positives and true negatives) among the total number of cases examined. It gives an overall picture of how often the model is correct across all classes. Precision is used to measure the proportion of true positive results in all positive predictions. It is a measure of the accuracy of the

positive predictions made by the model, indicating how reliable the model is when it predicts a positive class. On the other hand, recall is the proportion of true positive results in all actual positives. This metric indicates the ability of the model to find all relevant cases within a dataset. Lastly, F1 score is the harmonic mean of precision and recall, providing a single metric to measure the balance between them [6]. Using these metrics will allow for a comprehensive assessment of the classification models, resulting in a better grasp of their complexity, accuracy, efficiency, advantages and disadvantages.

Through detailed analysis, this study aims to not only enhance the tools available for app development but also to enrich our understanding of how automated systems can be effectively implemented to manage user feedback. By advancing the field of user review classification, this research hopes to pave the way for more sophisticated approaches to handling user feedback, ultimately leading to better digital products and user experiences.

1.2 Aims and Objectives

The black bullet points below represent the aims and objectives of this study. The white bullet points represent the criteria for success for the black bullet points above them.

- Collect different datasets which consist of real reviews. Datasets may differ in terms of data size and weight balance of the classes. Analyze the content and properties of the datasets and perform preprocessing.
    - First and foremost, the dataset must be English, organized, clean, relevant to the project and consistent. Data size should not be deficient or excessive. Properties of the datasets are analyzed and relevant content is stored. Data preprocessing must be accurate and well organized.

- Design and implement three models CNN, DistilBERT and XLNet for the classification of user reviews. Complete the two main experiments (50-25-25 split and K-Fold Cross Validation) using the models.
    - The design of the models must be compatible with multiclass text classification. Model complexity should not be effortless or rigorous to the computer. Complete the two experiments successfully. Perform the experiments with different hyperparameters. Analyze the results using the evaluation metrics accuracy, precision, recall and F1 score.

- Perform an analysis of the experiment results and store the corresponding values for future comparison and evaluation. Using the results, compare the performance of the models.
    - Evaluation metrics must be analyzed accurately and deeply. Along with the evaluation metrics, the hyperparameters used during the experiments must be stored. statistical measures and visualizations such as tables, charts and histograms should be used to gain a better understanding of the model's performance.

1.3 Report structure

This report contains six chapters:

• Chapter 1 Introduces the project, outlines the motivation for the project, and states the aims and objectives.
• Chapter 2 provides the reader with sufficient knowledge to understand the results, along with information about related work in the field.
• Chapter 3 Describes the methodology used in the project, including data collection, model selection and experimental setup.
• Chapter 4 presents the results of the project, performs Quantitative and Qualitative analysis of the results.
• Chapter 5 compares the project with related work as well as reflections and limitations on the project
• Chapter 6 conclusion of the report is presented, providing insights into the possible future direction of the project.

2 Background Information

This background section provides the necessary information to understand the study's results and helps to understand the significance of the findings within the broader field.

2.1 Background for app reviews

The app industry is experiencing rapid growth, both in terms of the number of apps and the number of users. With nearly 9 million applications available on platforms like the Google Play Store and the App Store, it is clearly visible that applications are everywhere. From the moment we wake up, to the daily activities we encounter and by the time we sleep. There is an application for everything at anywhere for anyone. As a result, there has been a significant increase in the volume of app reviews being generated through app stores such as Google Play Store, iTunes App Store, and Microsoft Store. These app stores not only serve as platforms for software distribution but also provide a space for users to share their feedback in the form of ratings and reviews. This shift in user behavior, from passive consumers to active contributors, allows for the sharing of real experiences that can be valuable for other users when deciding whether or not to use a particular app [7].

Users have the option to assign a rating to your application using a rating system that ranges from one to five stars. The ratings provided by individuals are then used to calculate the overall rating of your app, which is showcased on your product page and in search results. Written reviews also allow users to provide more in-depth information about their personal experience with your app [8].

These reviews function as a means of communication between developers and users, enabling users to provide important information to guide app developers in executing various software maintenance and evolution tasks, including the implementation of novel features, bug fixing, or the enhancement of current features or functionalities. App developers devote significant effort to gathering and leveraging user feedback to enhance user satisfaction [9].

2.2 Machine Learning Models

Machine Learning is a subfield of artificial intelligence that allows a system to learn from data and improve its performance over time without being programmed explicitly. Machine learning involves algorithms that can process, analyze, and learn from data, allowing machines to behave more wisely by performing particular processes [10]. Machine learning’s ultimate aim is for a computer to learn automatically and improve its capacity to adapt to new data in its absence. Machine learning has been

effective in many data analysis operations such as anomaly detection, natural language processing, image classification, decision-making processes and has been applicable in areas ranging from healthcare, finance, engineering, and security, among others [11].

Machine learning models are sophisticated mathematical constructs that enable computers to make decisions or predictions based on the analysis of data. These models differ from traditional algorithms that require specific instructions to operate. Instead, machine learning models modify their parameters and enhance their capabilities as they process more data. These models are adept at identifying patterns, relationships, and structures within the data, which enables them to make informed predictions and decisions driven by the data itself, rather than by explicit programming [12].

Some examples of traditional machine learning models include Logistic Regression, Decision Trees, Support Vector Machines (SVM), Naive Bayes, and k-Nearest Neighbors (k-NN). These models form the foundation of numerous applications, learning from past data to facilitate a wide range of functions. They are generally categorized into three main types: supervised, unsupervised, and semi-supervised learning, each designed to meet different requirements based on the data's nature and the specific challenges of the task [13].

2.3 Supervised Learning

Supervised learning is a method of machine learning where models are taught using a dataset that has been labeled. This means that each example in the training set has been matched with the correct output [14]. The dataset used during the project consisted of both context and labels for the context. The model learns to make predictions or decisions based on the input data by recognizing patterns, connections, and structures in the labeled information. Throughout the training process, adjustments are made to the model's parameters whenever the predicted labels are different from the actual labels.

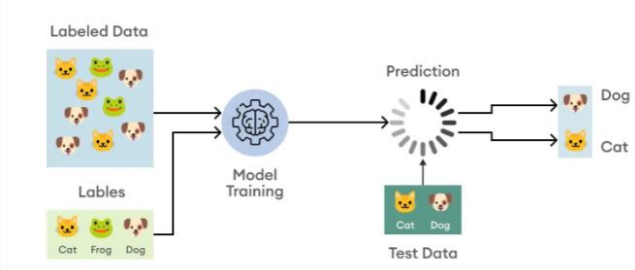


Figure 2.1: Visual Representation of Supervised Learning

When Figure 2.1 is observed, it can be seen that the purpose of supervised learning is to create a model that can make accurate predictions on new, unseen data that follows the same patterns as the training set.

Common tasks in supervised learning include classification, where the model uses an algorithm to accurately assign test data into specific categories, and regression, where the model predicts a continuous result [15]. For this project, supervised learning is used for classification task.

Supervised learning utilizes different classification algorithms, with Support Vector Machines (SVM) and Naive Bayes being the most commonly used ones. The main goal of classification tasks is to produce an output value that corresponds to a finite number of options or categories [16]. The data in this project can be classified into four categories: bug reports, user experiences, feature requests, and rating.

2.4 Neural Networks

Neural networks, or artificial neural networks (ANNs), are a subset of machine learning inspired by the structure and function of the human brain [17]. The networks use interconnected nodes or neurons arranged in layers, similar to the structure of the biological nervous systems.

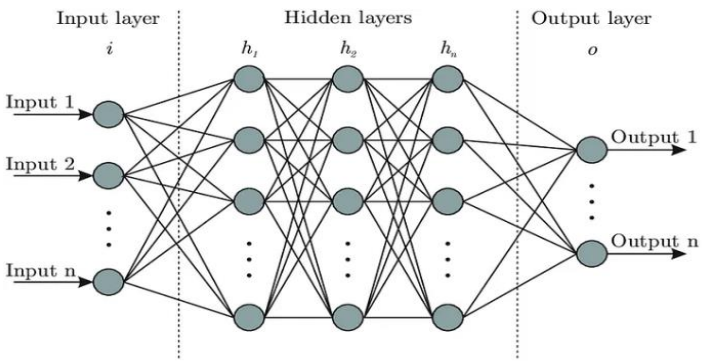


Figure 2.2: Architecture of Neural Networks

Figure 2.2 explains the architecture of a typical neural network. A neural network is composed of different layers of nodes known as artificial neurons. These layers include an input layer, one or more hidden layers, and an output layer. Each node is connected to others and has its own weight and threshold. Each neuron receives input, and processes it according to its internal state and a set of parameters [18]. When a node's output exceeds the threshold value, it becomes activated and passes data to the next layer. If the output does not meet the threshold, data is not transmitted to the next layer.

Neural networks are capable of learning complex patterns and relationships within large sets of data, making them highly effective for a wide range of applications such as image recognition, computer vision, speech recognition and natural language processing (NLP) [19]. Using neural networks is essential

for this project as all three models used in the project CNN, DistilBERT and XLNet are built on the principles of neural networks.

2.5 Deep Learning Models

Deep learning is a subset of machine learning that uses neural networks. These multi layered deep neural networks, which are inspired by the structure of human brain, are used by the deep learning models to perform classification tasks and recognize patterns in photos, text, audio and other various data [20]. Unlike traditional neural networks, which might have just a few layers, deep learning networks can have hundreds or thousands of layers, each capable of learning different features at various levels.

This allows deep learning models to handle large volumes of data and perform tasks such as image and speech recognition, natural language processing, and predictive analytics with high accuracy. Because deep learning models achieve promising results for such important tasks, deep learning is used in a variety of fields. Some examples are autonomous vehicles, robotics, medical applications, financial services, virtual assistance, and predictive analysis [21].

Deep learning models, even though they are a category of machine learning algorithms, differ from traditional machine learning. The key difference between machine learning and deep learning lies in their approach to feature extraction: machine learning algorithms often require manual selection of features from data, whereas deep learning models automatically discover and learn the features directly from raw data [22]. This allows deep learning models to handle vast amounts of unstructured data and perform more complex tasks, like image and speech recognition, more effectively than traditional machine learning models. However, deep learning models typically require larger datasets and more computational power to train [23].

In the last few decades, a large number of deep learning models for text classification have been suggested. Some of the examples are RNN, CNN, attention and GNN [24]. Since this project is about multi-class text classification, a deep learning model that is efficient and accurate such as Convolutional Neural Network (CNN) will be used.

2.6 Convolutional Neural Network (CNN)

Convolutional Neural Network, also known as CNN, is an area of deep learning specializing in pattern recognition. The key feature that makes CNN different from regular neural networks is the architecture difference [25]. Typically, Convolutional Neural Networks consist of three main layers. They

are Convolutional, Pooling, and fully connected (FC) layer.

A convolutional layer is a core building block of Convolutional Neural Networks (CNNs), designed to automatically and adaptively learn spatial hierarchies of features from input images or sequences. This layer applies a set of learnable filters to the input. These filters slide over the input, computing dot products to produce 2D activation maps that highlight features such as edges, textures, and more complex patterns at various depths of the network [26].

A pooling layer in CNNs serves to downsize the input's spatial dimensions, often using max pooling to select the maximum value from non-overlapping regions. This reduces the network's computational load and parameters, helps prevent overfitting, and makes feature detection more resilient to variations in scale and orientation [26].

Finally, A fully connected (dense) layer connects every neuron in the layer to every neuron in the previous layer, integrating learned features for predictions or classifications. It applies weighted sums and an activation function to transform inputs into outputs, crucial for tasks like classification following convolutional and pooling layers in CNNs [26].

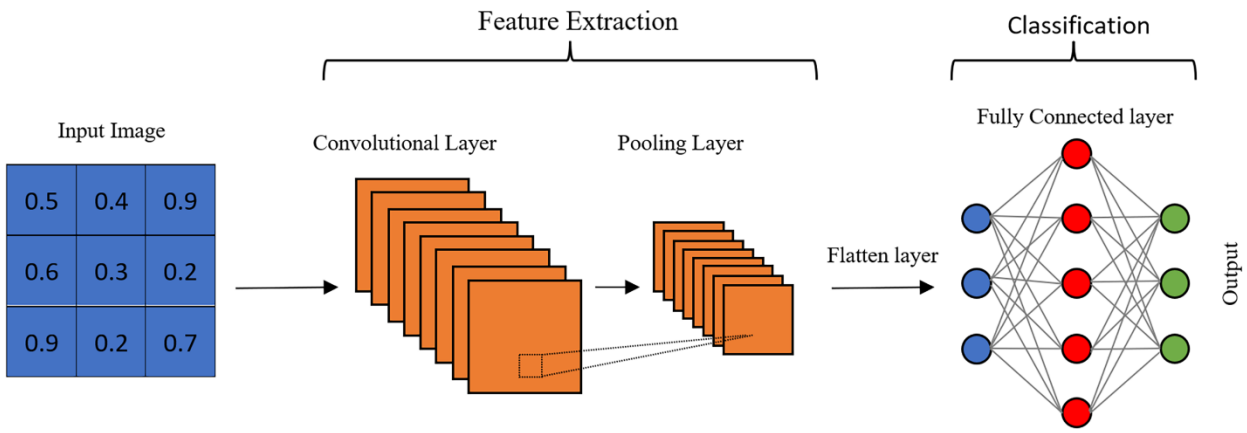


Figure 2.3: Visualization of typical CNN

Figure 2.3 visualizes the explained architecture of CNN. The architecture of CNNs enables them to capture important features like edges, textures, and patterns in the early layers, and more complex objects and semantics in deeper layers, making them particularly effective for tasks such as image and video recognition, image classification, and object detection, among others.

Although image analysis has been the most widespread use of CNNs, it can also be used for other types of data analysis and classification problems [27]. For this project, CNN is used for multiclass text classification. It is pretty natural to say how can CNN be used for text classification when it is known to be specialized in image analysis. Here is an overview of how CNN is used for this project.

CNN can be adapted for text classification by treating text as sequential data, similar to spatial data in images. Text is first converted into numerical form through embedding, transforming words into dense vectors. CNNs then apply convolutional filters to these embeddings to capture patterns within windows of words, effectively learning semantic features. Following convolution, pooling layers reduce dimensionality and extract the most salient features, which are then flattened and fed into fully connected layers for the classification task [28]. A more detailed explanation of how this model is used during the project will be explained in further chapters.

2.7 Large Language Models

Large Language models, also known as LLMs, are a subset of deep learning. They are based on transformer architecture. Transformers utilize a mechanism known as self-attention or the attention mechanism, which allows the model to weigh the importance of different words within a sentence, regardless of their positional distance from each other. LLMs have been trained on enormous volumes of textual data, and have the capability to produce text that is similar to human language. They can also accurately respond to queries, provide answers, and perform other language-based tasks with great precision [29]. In transformer architecture, commonly used in large language models (LLMs), the concepts of encoder and decoder are fundamental components that process and generate sequences, respectively.

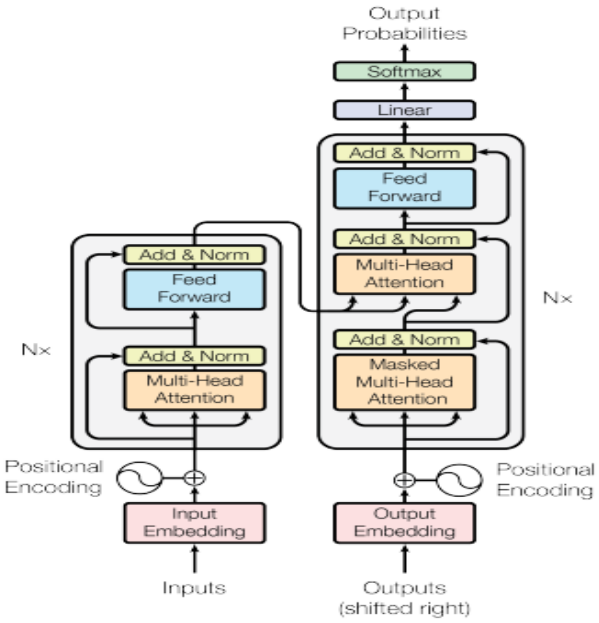


Figure 2.4: The Transformer Model Architecture

Figure 2.4 explains the architecture of the Transformer model and how the encoder-decoder works. The encoder's role in transformer architecture is to process the input sequence, transforming it into a continuous representation that holds the context of each word or token relative to the entire sequence. It consists of multiple layers, each containing self-attention mechanisms and feed-forward neural networks. These layers work together to capture the complexities and dependencies in the input data, allowing the model to understand the meaning and relationships between the elements of the input [30].

The decoder, on the other hand, is responsible for generating the output sequence from the encoded information. Like the encoder, it comprises multiple layers that include self-attention mechanisms, feed-forward neural networks, and additionally, encoder-decoder attention mechanisms [31]. This last component allows the decoder to focus on different parts of the input sequence as it generates each token of the output, effectively translating the encoded context into a coherent and contextually relevant output sequence.

These models are trained for general purposes to solve common language models such as text classification, question answering, document summarization and text generation across multiple different industries [29]. The models can then be fine-tuned to solve specific problems using a relatively small size of field dataset.

There are a lot of benefits of using large language models. First and foremost, a single model can be used for different tasks with high accuracy. Large language models require minimal field training data when fine-tuned to solve specific problems. LLMs achieve high end results even with a limited amount of domain training data. Lastly, the performance of the models is continuously increasing with the increasing number of data and parameters [31]. ChatGPT is a great example of recent years, as compared to GPT-3's 17 gigabytes of data, GPT-4, the most recent iteration of OpenAI, has 45 terabytes of training data. As a result, GPT-4 can deliver significantly more accurate results than GPT-3. The subject LLM is critical for this project as two of the three models used during the project are large language models.

2.8 DistilBERT

Distilled Bidirectional Encoder Representations from Transformers also known as DistilBERT, is a distilled version of the original BERT model. It is designed to be smaller, faster, and more efficient while retaining most of the original model's language understanding capabilities. The architecture of DistilBERT is derived through a process known as knowledge distillation, where the smaller “student” model is trained to replicate the behavior of the larger, more complex “teacher” model which is BERT in this case [32].

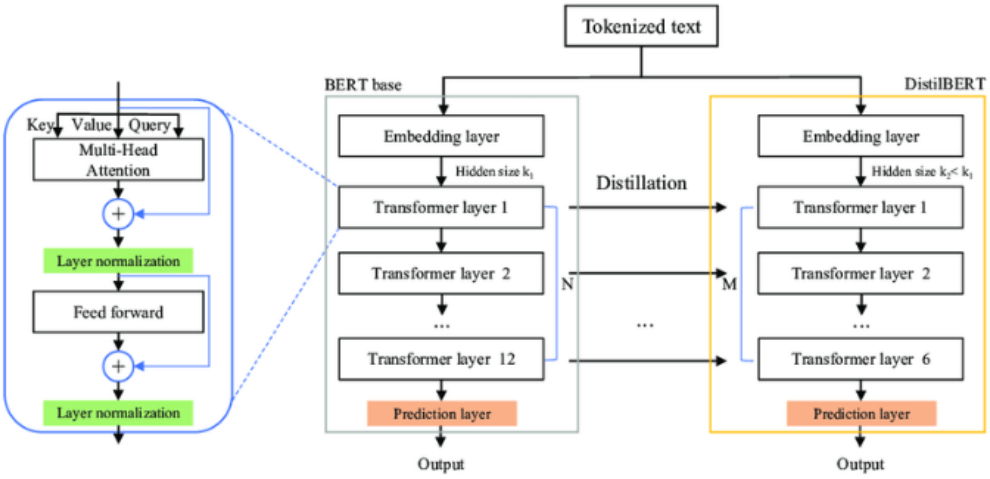


Figure 2.5: Architecture of DistilBERT with Comparison to BERT

After observing Figure 2.5, it can be said that the architectural design of DistilBERT follows the general framework of transformers, similar to BERT, but introduces modifications to increase efficiency. Specifically, the student model in DistilBERT reduces the number of transformer layers from twelve to six, and it also omits the token-type embeddings and the pooling layer that are present in the original BERT model. This refinement leads to a reduction in the total number of parameters from 110 million in the original BERT to 66 million in DistilBERT, marking a decrease of 40%. Nonetheless, despite this substantial reduction in size and complexity, DistilBERT successfully retains approximately 97% of the language comprehension capabilities of the original BERT model, thanks to its preserved transformer architecture [33].

Performance evaluations of DistilBERT highlight its ability to process tasks 60% faster than its predecessor while nearly matching BERT's performance in language understanding. The combination of reduced model size, increased processing speed, and high accuracy makes DistilBERT particularly advantageous for applications requiring efficient language processing capabilities [34]. These attributes have significantly influenced the decision-making process regarding which models to utilize in various machine learning projects, particularly those involving natural language processing tasks. The efficiency and effectiveness of DistilBERT make it a preferred choice for developers and researchers aiming to integrate robust language understanding capabilities into their systems without the computational and resource overhead required by the full BERT model.

2.9 XLNet

XLNet is an advanced natural language processing (NLP) model developed by researchers at Google Brain and Carnegie Mellon University, which outperforms traditional autoregressive and autoencoding models like BERT and GPT through its unique approach to language modeling. XLNet employs permutation language modeling, a technique that predicts each word in a sentence in a randomly

determined order, thereby learning the context of words in all possible configurations [35]. This method significantly enhances the model's ability to generalize across various NLP tasks by capturing bidirectional context without being limited to a fixed sequence order.

XLNet's architecture is particularly notable for incorporating features from Transformer-XL, which provides an extended contextual understanding beyond fixed-length contexts. This is crucial for the model's performance on complex NLP tasks, where a deep understanding of extended narratives or discussions is necessary. Through its permutation-based training and ability to learn diverse patterns in language, XLNet stands out as a highly effective model for a wide range of NLP applications, delivering superior performance in understanding and generating language [36].

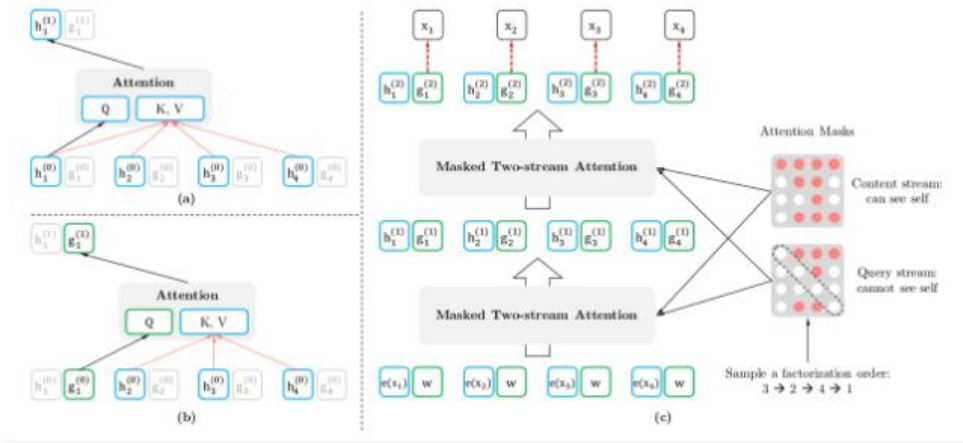


Figure 2.6: Architecture of XLNet

Figure 2.6 visualizes and explains the architecture of XLNet. XLNet introduces a two-stream self-attention mechanism that consists of a content stream and a query stream. The content stream captures the context of all tokens, whereas the query stream is designed for predicting a particular token. Unlike traditional language models that predict the next token in a sequence, XLNet uses permutations of the input sequence to predict tokens [37]. This means that for any given token, the model considers all possible previous tokens in various orders, thus learning a more generalized representation of the language.

2.10 K-Fold Cross Validation

When doing a project that involves machine learning/ NLP, performance evaluation is just as important as designing and training models. K-fold cross-validation (KCV) is a method which is used to evaluate the performance of a trained model on a limited data sample. The method uses 3 key steps while performing the model evaluation [38].

Firstly, the dataset is divided into 'k' number of equally sized subsets or folds. Each one of the k subsets serves as the validation set to evaluate the model. Later the remaining k-1 folds are combined to form the training set which the model uses for training. This procedure is repeated 'k' times, with each of the k folds used exactly once as the validation set. Lastly, after completing all k iterations, the average performance metric across all k folds is calculated to provide an overall estimation of the model's predictive performance [39].

KCV is one of the most popular methods used by practitioners for classifier error estimation and model selection [40]. The reason behind that statement is the number of advantages KCV offers. First of all, the method offers more reliable estimates. The average performance metric computed across all k iterations offers a more reliable estimate of the model's performance on unseen data compared to a single train-test split. This is due to the reduction in variance associated with the evaluation metric. K-fold cross-validation exposes the model to different subsets of the data during each fold. This helps prevent overfitting, as the model learns from diverse samples. Also, the method can be applied to any model, making it a versatile tool for comparing the performance of different modeling approaches on the same dataset. KCV was used for this project as a versatile and reliable performance metric was required for testing three different models [41]. However, the method is not perfect. The main drawback of this method is the increased computational cost. Training and evaluating the model k times can be time-consuming. This disadvantage will be explained in detail in further chapters.

2.11 Related Work

In the context of app development, the significance of user reviews has been prominently discussed in recent research. For instance, a study by Maalej et al. [42] delved into the automated classification of app reviews, highlighting how user feedback in app stores can be a rich resource for developers. This feedback often includes direct user responses on app functionalities, suggestions for improvements, and reports of bugs, which are crucial for iterative development processes. Their research demonstrated the use of probabilistic techniques and natural language processing to categorize reviews into specific types such as bug reports, feature requests, and user experiences. The findings emphasized that integrating text classification with review metadata significantly enhances the precision and recall of categorizing reviews. Moreover, the study illustrated that multi-type classifiers, when combined with advanced natural language processing, could adaptively improve the sorting of user feedback, thereby directly influencing app

enhancements and prioritization of developer responses.

In recent research by Guzman, El-Halaby, and Bruegge [43], the importance of classifying app reviews for software evolution is thoroughly explored. Their study proposes an innovative taxonomy for categorizing user feedback into categories that are crucial for the evolution of software, recognizing the substantial challenges developers face in analyzing user feedback manually due to the sheer volume and often unstructured nature of user reviews. They highlight how ensemble methods, which combine the strengths of individual machine learning classifiers, can significantly enhance the accuracy and efficiency of review classification systems. This approach proves particularly effective in their experiments, outperforming single classifier systems in precision and recall, thus providing a more nuanced and actionable analysis of user feedback. The work underscores the critical need for advanced machine learning techniques to manage and utilize user feedback effectively. By automating the classification of app reviews into defined categories, developers can quickly and accurately gauge user sentiment and prioritize updates or fixes, which is essential for maintaining the relevance and functionality of their applications in a competitive market

Chapter 3: Research Methodology

My project has followed a systematical approach consisting of the following steps:

- Step 1: Data Collection
- Step 2: Data Preprocessing
- Step 3: Defining Classification Tasks
- Step 4: Model Selection
- Step 5: Model Design/Implementation
- Step 6: Model Training
- Step 7: Model Evaluation and Performance Measures

3.1 Collection of Data

Acquiring or discovering the dataset is the primary and crucial step in this project as the quality and quantity of data directly impact the model's performance. For this project, the dataset produced by Drmingler [44] was used., which includes two distinct CSV files known as "augmented" and "dataset_guzman_labelled".

The "augmented" file has a total of 7311 reviews that are distributed into 1806 bug reports, 2117 user experiences, 1760 feature requests, and 1628 ratings. Similarly, the second file, "dataset_guzman_labelled," includes a total of 6159 reviews that are distributed into 990 bug reports, 2518 user experiences, 404 feature requests, and 1628 ratings. Both datasets gather reviews from various applications like WhatsApp, Tripadvisor, Evernote, Picsart, Pinterest, and Dropbox.

These datasets were a good fit for the project as the source of the reviews are well-known global companies and the four classes of the reviews directly match the description of the project. However, not all data from these datasets are used in this project. The datasets will be modified during the data preprocessing phase which will be explained in the next section.

3.2 Data Preprocessing

Data preprocessing is a crucial initial step in NLP that involves cleaning and transforming raw textual data into a format that can be efficiently and effectively analyzed by NLP algorithms and models.

3.2.1 Dataset Modification

The first data preprocessing task was modifying the original datasets. Only the necessary columns (reviews and task/label) were kept and the rest of the columns were removed from the datasets. A Python script is written and used to randomly select 2000 reviews from each category across the two original datasets and create a new single dataset consisting of 8000 balanced reviews.

Using a balanced dataset is critical for NLP tasks for several reasons. Using an unbalanced dataset can cause the models to be biased to majority classes which worsens the model’s performance. Also in unbalanced datasets, traditional evaluation metrics like accuracy can be misleading because they might reflect the underlying class distribution rather than the model's ability to learn from the data. Lastly, having a balanced dataset allows for more experimentation and comparison of different models, preprocessing techniques, or feature engineering strategies. This is crucial as the project’s objective is to compare 3 different models using different experiments.

3.2.2 Different Preprocessing Techniques

Multiple different preprocessing techniques are popular amongst NLP tasks such as case folding, stop word removal, stemming and lemmatization. However, none of these techniques were used during the

project because, for large language models and similar architectures, the necessity of this preprocessing is not as critical as it is for classical NLP models. This is because LLMs, especially those based on Transformer architectures, are designed to understand and generate human-like text by capturing the context of words in sentences. Removing stop words or altering word forms through stemming or lemmatization can sometimes remove subtle meanings of the text that could be important for understanding the context fully.

Additionally, LLMs are trained on vast amounts of text data from diverse sources, which helps them generalize well across different linguistic structures, vocabularies, and styles without the need for specific preprocessing steps. They are designed to understand language in its natural form, including the use of stop words that play a role in sentence structure and meaning.

3.2.3 Pre-Tokenization Processes

Even though traditional preprocessing techniques were not necessary for this project, there is still preprocessing done before the tokenization. First of all, the acronyms of the class labels are rewritten into the label column. For example, feature request is written instead of FR, the same for bug report/PD, rating/RT, and user experience/UE. Then two new columns “count” and “Encoded_text” are added to the dataset as it can be seen in Figure 3.1. The “Count” column counts the number of words that a review has for each row. This information is important during the installation of the tokenizer. “Encoded_text” assigns a unique integer for each class in the label column. For this instance, bug report is 0, feature request is 1, rating is 2 and user experience is 3. This process is critical during the model evaluation and prediction on the test data. Finally, using the count column, a graph is created which can be seen in Figure 3.2. This graph shows the number of reviews and the count of words in each review which is used during the tokenization stage.

	review	label	count	encoded_text
0	Seems more transparent & meaningful issues.	rating	6	2
1	The best app for editing texts	rating	6	2
2	It seems like it would be a great addition to ...	feature request	27	1
3	Its really fun and any age can play it and enj...	user experience	20	3
4	Hey whatsApp , atomic_number_53 am experiancin...	bug report	43	0

Figure 3.1: Dataset Representation after Preprocessing

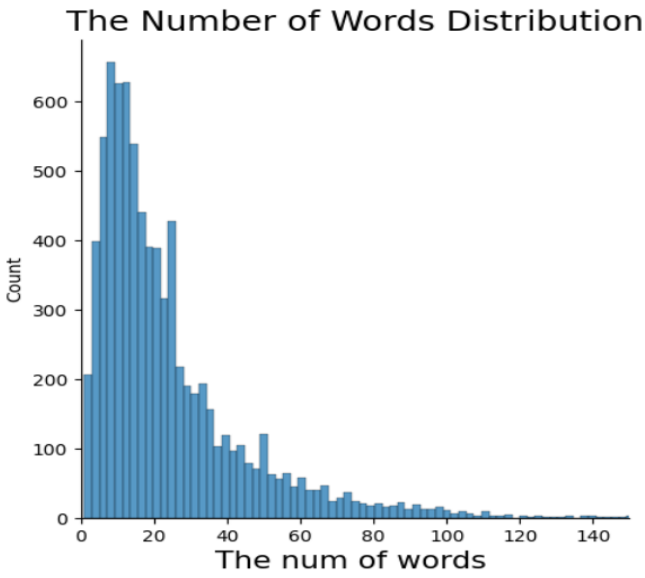


Figure 3.2: The Number of Word Distribution Graph

3.2.4 Tokenization

In natural language processing tasks, tokenization is a fundamental process of breaking down text into smaller units, called tokens. These tokens can be words, phrases, symbols, or other meaningful elements. Tokenization is one of the most important processes before starting the implementation and training of the models. These models require numerical input, so text must be converted into a format that models can understand. Figure 3.3 presents an example of text converted to a format that models can understand. During the project, different tokenizers are used. For DistilBERT and XLNet, DistilBertTokenizer and XLNetTokenizer are used from the transformers library. These tokenizers are specifically designed for the models. However, for the CNN model, a tokenizer from Keras library was used. For all 3 tokenizers, the number of word distribution graph is used to determine the maximum length of the input sequence that will be tokenized. After inspecting the graph, the maximum length was determined to be 128 as it contains more than 95% of the sequences. The input sequences are then padded to ensure that they are all the same length. After this process, the input was ready to be fed to the models.

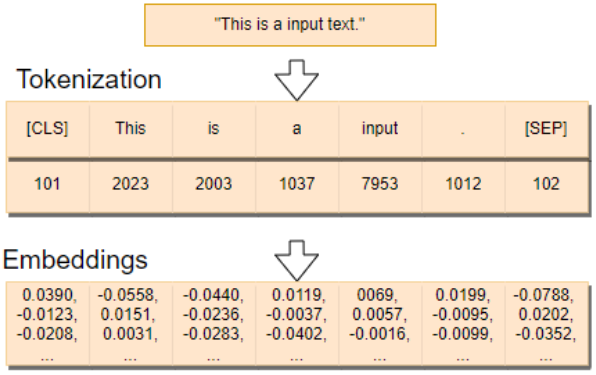


Figure 3.3: Visual Representation of Tokenization

3.3 Defining Classification Tasks

In this project, two multiclass text classification tasks were used to evaluate and compare the performance of 3 different models. The first task was to use a 50-25-25 split on the data which the models use 50% of the data for training, 25% for validation and 25% for testing. Using a fixed training and testing split to compare different models offers a straightforward, consistent, and efficient method for evaluating their performance on a text classification task. This approach ensures that all models are trained and tested under the same conditions, allowing for objective comparisons based on their ability to generalize from training to unseen data.

On the other hand, the second task uses the K-Fold cross validation method. This method divides the dataset into k distinct subsets (or "folds"), then iteratively trains the model k times, each time using a different fold as the validation set and the remaining data as the training set. This process results in each data point being used both for training and validation across the folds. The final performance metric is the average of the performance across all k folds, offering a more robust and reliable estimate of the model's ability to generalize to unseen data.

Even though both tasks use different methods to train and test the models, the main objective is the same. The aim is to evaluate, compare then find the most accurate and efficient model that can identify if a review represents bug report, user experience, feature request or rating.

3.4 Model Selection

For the project, several machine learning, deep learning, and large language models were accessible for use. Out of all the models, CNN, DistilBERT, and XLNet were chosen to complete the project. The following are the key factors that led to the selection of each model.

Convolutional Neural Networks are effective at extracting hierarchical features from spatial data. When applied to text, convolutions can effectively capture local dependencies and patterns such as n-gram features within sentences, making CNNs reliable at identifying key phrases indicative of each category. By stacking multiple convolutional layers, CNNs can learn hierarchical features, which can be beneficial for understanding the semantics of different review types. Moreover, CNNs can be relatively lightweight and fast compared to some deep language models, offering quicker training and inference times. A seminal paper by Yoon Kim in 2014 titled "Convolutional Neural Networks for Sentence Classification" demonstrated how simple CNN architectures could achieve excellent performance on multiple text

classification benchmarks [45].

DistilBERT, a distilled version of BERT is excellent for multiclass text classification tasks where efficiency and high performance are desired as the model can handle long sequences and capture complex relationships between words. DistilBERT captures bidirectional context, which is crucial for understanding the meaning of words in context. Despite being the smaller version of BERT, the model retains most of the original model's ability to understand the context and semantics of text. Its understanding of language nuances makes it highly effective for accurately classifying text into specific categories based on subtle differences in meaning. DistilBERT provides a good balance between computational efficiency and model performance. It's suitable for environments where resources are limited but a high understanding of language context is required. DistilBERT's efficiency and effectiveness were highlighted through its performance on the GLUE benchmark, a collection of diverse NLP tasks including text classification, sentiment analysis, and question answering [33].

XLNet, based on Transformer XL, captures bidirectional context by maximizing the likelihood over all permutations of factorization order. This allows XLNet to understand complex sentence structures and the context better, which is crucial for accurately classifying app reviews where context can significantly change the meaning. XLNet can learn hierarchical representations, capturing both local and global dependencies in text data. The paper" XLNet: Generalized Autoregressive Pretraining for Language Understanding" by Yang et al., demonstrated state-of-the-art performance of XLNet on several NLP benchmarks, surpassing BERT and other models at the time of its release. Its performance on benchmarks like SQuAD for question answering and GLUE for various NLP tasks underscored its superior understanding of language context and its ability to handle complex linguistic structures [35].

3.5 Programming Environment and Libraries

For this project, multiple different platforms and libraries were used. First of all, Python3 was used as the programming language as it has a rich ecosystem of libraries and frameworks designed specifically for AI and NLP tasks. Libraries such as TensorFlow and PyTorch are perfect examples. Most of the time, Google Collab was used as the primary coding platform because of the access to free T4 GPU. Using this free GPU helped with the computation time during the training and testing of the models. Moreover, Virtual Studio Code was used as the secondary platform to form the structure of the project.

Pandas and Numpy libraries were used for data manipulation, analysis, and numerical computing. Tensorflow and Keras, which are built on top of Tensorflow were used to build the CNN model, train and test the models as well as provide a tokenizer. Hugging Face library was used to access the Transformer models such as DistilBERT and XLNet. Matplotlib and Seaborn were used for plotting graphs during the preprocessing. Itertools was used during the hyperparameter selection. Mlfow was used during the training of the models to keep track of the results. Scikit-learn is used for splitting the data, doing K-Fold Cross Validation and evaluating the performance of the models.

3.6 Model Design/Implementation

3.6.1 CNN Architecture Building

For the design of CNN architecture, Keras library was used. The fundamental layers such as Convolutional, Pooling, and fully connected (Dense) layer were implemented as well as some other layers. The first step of designing CNN architecture is to define the model type. In Keras, the model type can be defined as a sequential model. The use of Sequential() function initializes a linear stack of layers, creating the foundation for a CNN model. This setup allows layers to be added in sequence, with each layer feeding its output to the next layer in a straightforward, linear manner. Here are the implementations of the CNN layers.

Embedding Layer: The Embedding layer transforms integer-encoded vocabulary into dense vectors of fixed size (embedding_dim). This layer is crucial for processing text data, as it converts word indices into embeddings that capture semantic relationships between words. The parameters input_dim=max_words and input_length=max_len specify the size of the vocabulary and the length of input sequences, respectively. This representation serves as the input to the convolutional layers, enabling them to learn from the semantic content of the text.

Convolutional 1D Layer: The Conv1D layer applies one-dimensional convolution operations to the embedded word vectors. It uses a specified number of filters (num_filters) and a kernel size (kernel_size), which determines the window size over which the convolution is applied. The relu (rectified linear unit) activation function introduces non-linearity, allowing the model to learn complex patterns. This layer is adept at detecting local patterns within the sequence, such as phrases or sequences of words that carry specific meanings relevant to the classification task.

Global Max Pooling 1D Layer: Following convolution, the GlobalMaxPooling1D() layer reduces the dimensionality of the data by applying max pooling over the time dimension. This operation selects the highest activation across the entire sequence for each filter, summarizing the most significant features

detected by the convolutional layers. This step helps to reduce computational complexity and mitigates the risk of overfitting by focusing on the most important features.

Dropout Layer: The Dropout layer randomly sets a fraction (dropout_rate) of the input units to 0 at each update during training, which helps prevent overfitting. This layer provides a form of regularization by forcing the model to learn more robust features that are not reliant on any small subset of the neurons.

Dense Layer with Softmax Activation: Finally, the Dense layer serves as the output layer of the model. It maps the pooled features to the desired number of classes (4 in this case. The softmax activation function converts the logits to probabilities for each class, facilitating multi-class classification.

3.6.2 Implementing DistilBERT Model

The used DistilBERT model was imported from the Transformers library by HuggingFace. Using a single line such as “TFDistilBertForSequenceClassification.from_pretrained('distilbert-base-uncased', num_labels=4)”, DistilBERT model architecture can be imported and used. Unlike the CNN model, LLMs don’t require to build an architecture as they are pre-trained and openly available. The model is initialized with weights from 'distilbert-base-uncased', which means it uses a base version of DistilBERT that has been pre-trained on a large corpus of English text in a case-insensitive manner. This pre-training on a general language understanding task allows DistilBERT to understand basic language structures and semantics before being fine-tuned on a specific sequence classification task.

On top of the DistilBERT model, a sequence classification head is added, consisting of a single dense layer that maps the representation of the input sequence from DistilBERT to the desired number of classes which in this case is 4. This layer takes the pooled output of the DistilBERT model and produces logits for each of the four classes in the classification task.

3.6.3 Implementing XLNet Model

Similarly, the XLNet model was imported from the Transformers library by HuggingFace as well. Using a single line such as “TFXLNetForSequenceClassification.from_pretrained('xlnet-base-cased', num_labels=4)” XLNet model architecture can be imported and used. Like the DistilBERT model, XLNet don’t require to build an architecture as it is pre-trained and openly available

The “from_pretrained” method initializes the model with weights from a pre-trained version of XLNet, specifically 'xlnet-base-cased'. This means the model has already been trained on a large corpus of

text data and has learned a rich understanding of language, including syntax, semantics, and context. The pre-trained model can then be fine-tuned on a specific dataset or task, such as sequence classification, leveraging the general language knowledge it has acquired.

The 'xlnet-base-cased' section specifies the use of the "base" version of XLNet, which is a smaller, more computationally efficient version compared to the "large" variant, but still powerful enough for a wide range of tasks. The "cased" part indicates that the model retains the case sensitivity of the input text, distinguishing between uppercase and lowercase letters, which can be crucial for understanding the meaning of words and sentences in certain contexts.

On top of the XLNet architecture, a sequence classification head is added. This consists of a dense layer with a softmax activation function that takes the output from the XLNet model and produces a probability distribution over the four possible classes. The model learns to map the complex, contextualized representations generated by XLNet to specific class labels, enabling accurate classification of input sequences based on their content.

3.7 Model Training

This section explains the process of all 3 models training on 50-25-25 split and K-Fold Cross Validation.

3.7.1 Training the CNN model

Method 1: First of all, the method starts by splitting the dataset into 50% training, 25% validation and 25% test sets. Then, using both the number of word distribution graph and the Keras tokenizer, the 3 sets are tokenized to be fed to the model. After the model architecture was completed, 4 different hyperparameters were selected to experiment on. These hyperparameters were the output dimension of the embedding layer, the number of filters in the convolutional layer, the kernel size in the convolutional layer and the dropout rate. For the main training loop, Mlflow was imported. MLflow is an open-source platform designed for the machine learning lifecycle, including experimentation, reproducibility, and deployment. It helps manage the end-to-end machine learning development process, enabling users to track experiments, package code into reproducible runs, and share findings. Before any training on the model, Mlflow experiment was initialized and hyperparameters were logged into the experiment. Then, the model started to train on every hyperparameter configuration on 5 epochs with a batch size 8, Adam optimizer with a learning rate of 1e-5 and Sparse Categorical Crossentropy as the loss function. After the training was completed, training accuracy/loss and validation accuracy/loss values were stored in Mlflow. Later, the model is evaluated and test loss/accuracy is stored in Mlflow. Finally, the model predicts the test

reviews into labels. The predicted labels are then compared with the correct labels. The evaluation metrics accuracy, precision, recall and f1 score are calculated and stored in Mlflow. Figure 3.4 shows the layout of Mlflow and how Mlflow stored the results for this experiment.

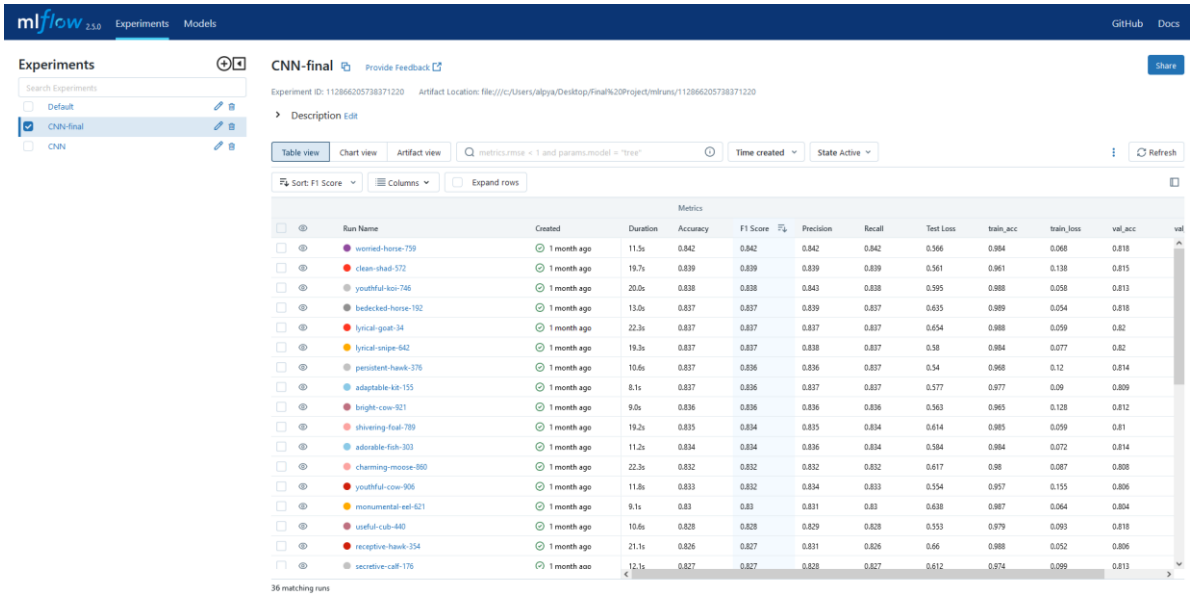


Figure 3.4: Layout of Mlflow

Method 2: Firstly, the k value in K-Fold Cross validation is set as 5. This means the dataset will be divided into five equal parts and the model will be trained on 4 parts and tested on the remaining part. This process will repeat 5 times until all parts are used for training and testing. At the start of the main loop, the dataset is split according to the fold index, and training and testing set is tokenized. Then, the best model from method 1 is identified using the tracking ability of Mlflow. The architecture and hyperparameters of the best model are implemented to the loop and the model is trained on 5 epochs with a batch size 8, Adam optimizer with a learning rate of 1e-5 and Sparse Categorical Crossentropy as the loss function. Similarly to the method, evaluate function is used to get the test loss/accuracy. Finally, the trained model is used to make predictions on the test data and metrics such as accuracy, precision, recall and f1 score are calculate to evaluate the model's performance.

3.7.2 Training DistilBERT and XLNet Models

Method 1: First of all, the method starts by splitting the dataset into 50% training, 25% validation and 25% test sets. Then each model uses the unique Tokenizer that was built for them by the transformers library and tokenizes the sets into tensors. The tokenized sets are then converted into TensorFlow datasets to be fed to the models. Both pre-trained models were imported from the transformers library. The models are then fine-tuned/trained on 5 epochs with a batch size 8, Adam optimizer with a learning rate of 1e-5

and Sparse Categorical Crossentropy as the loss function. Later, evaluate function is used to get the test loss/accuracy. Finally, fine-tuned models are used to make predictions on the test data and metrics such as accuracy, precision, recall and f1 score are calculated to evaluate each model's performance.

Method 2: KFold is initialized with 5 splits. This means the data will be divided into 5 parts, where each part will be used as a validation set while the rest will serve as the training set in each iteration of the cross-validation process. The main loop iterates over each fold, splitting the dataset into training and validation sets according to the indices The training and validation texts are tokenized using a unique tokenizer like method 1 to convert text into a format that the model can understand. The tokenization process involves truncating or padding texts to a uniform length of 128 and converting them to TensorFlow tensors. TensorFlow datasets are created from the tokenized texts and their corresponding labels. Then both pre-trained models are initiated from the transformers library. The models are fine-tuned/trained on 5 epochs with a batch size 8, Adam optimizer with a learning rate of 1e-5 and Sparse Categorical Crossentropy as the loss function, and accuracy as the metric. After training, the models are evaluated on the validation sets. The predictions are made using the models. Evaluation metrics such as accuracy, precision, recall, and F1 score are calculated using the true labels and the predicted labels for the validation sets.

3.8 Model Evaluation and Performance Measures

After the training of the models is completed, it is critical to evaluate the performance of the models on unseen data. This way the models can be analyzed and compared. For this project, 4 different metrics are used to evaluate the performance of the models. The metrics are accuracy, precision, recall, and f1 score. These metrics use values such as True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN) for calculations.

For a multiclass text classification, a true positive is an instance that is correctly classified as belonging to that class. A true negative refers to all instances that are correctly identified as not belonging to that class. A false positive occurs when an instance is incorrectly classified as belonging to that class when it does not. A false negative occurs when an instance that belongs to that class is incorrectly classified as not belonging to that class [46].

As for the metrics, accuracy measures the proportion of true results (both true positives and true negatives) among the total number of cases examined. It is the simplest metric to determine the overall correctness of the model [47].

$$Accuracy = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}}$$

Precision measures the accuracy of positive predictions. It is the ratio of correctly predicted positive observations to the total predicted positives. High precision relates to a low false positive rate [47].

$$Precision = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}$$

Recall, also known as sensitivity or the true positive rate, measures the proportion of actual positives that are correctly identified. It is the ratio of correctly predicted positive observations to all observations in actual class. High recall means most of the actual positives are correctly recognized [47].

$$Recall = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

The F1 score is a weighted average of precision and recall. It takes both false positives and false negatives into account. It is the harmonic mean of precision and recall for that class, providing a balance between the two [47].

$$F1\ score = \frac{2 * \mathrm{precision} * \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}$$

4. Result Analysis

This section analyses and compares the performance of all models according to the calculated results on unseen data.

4.1 Representation of Results

Table 4.1: Represents the values of evaluation metrics for all models in the 50-25-25 approach.

	CNN			DistilBERT			XLNet		
	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
Bug Report	0.82	0.85	0.83	0.77	0.94	0.85	0.86	0.90	0.88
Feature Request	0.87	0.80	0.84	0.86	0.79	0.83	0.85	0.84	0.84
Rating	0.83	0.84	0.83	0.86	0.81	0.84	0.81	0.85	0.83
User Experience	0.81	0.87	0.84	0.85	0.77	0.81	0.85	0.89	0.82

Figure 4.1: Represents the accuracy results of the models on unseen test data for the 50-25-25 approach

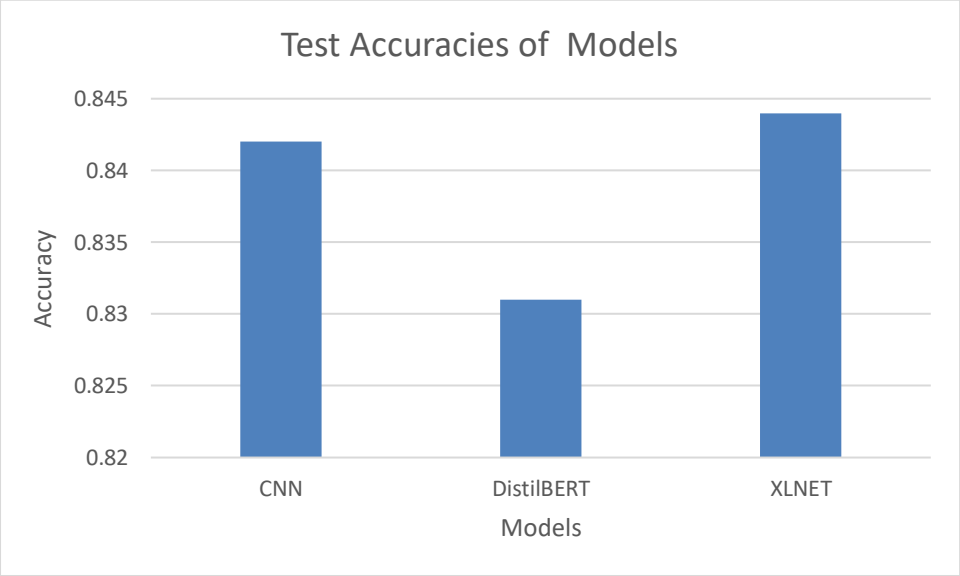
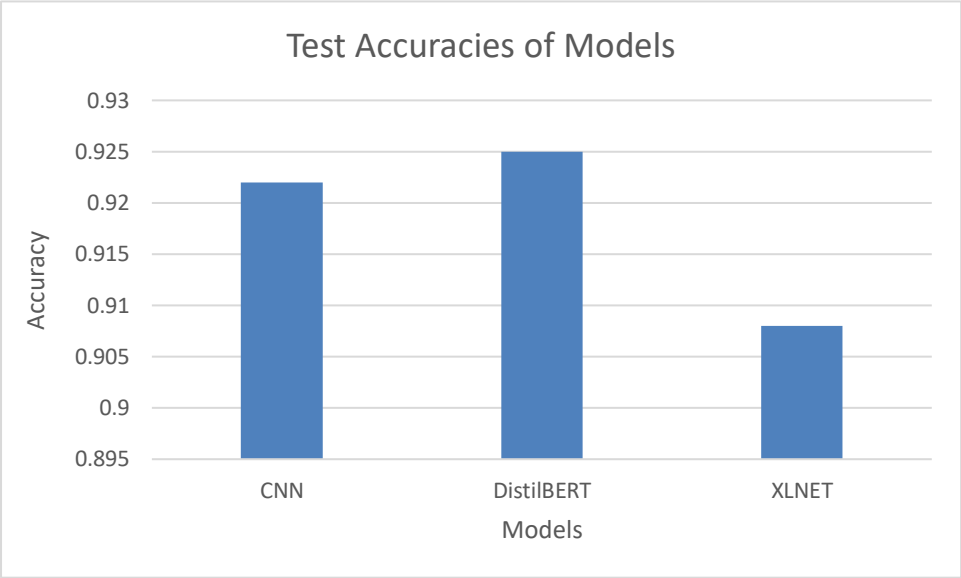


Table 4.2: Represents the values of evaluation metrics for all models in the K-Fold Cross Validation approach.

	CNN			DistilBERT			XLNet		
	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
Bug Report	0.94	0.92	0.93	0.94	0.97	0.95	0.90	0.97	0.94
Feature Request	0.94	0.92	0.93	0.95	0.90	0.93	0.91	0.90	0.91
Rating	0.91	0.92	0.92	0.89	0.95	0.92	0.96	0.84	0.89
User Experience	0.88	0.93	0.90	0.93	0.88	0.90	0.87	0.92	0.90

Figure 4.2: Represents the accuracy results of the models on unseen test data for the K-Fold Cross Validation approach



4.2 Analysis of the Results & Comparison of Models

Throughout the training and testing for both approaches, all hyperparameters such as batch size, epochs, loss function, optimizer, and learning rate are kept the same for every model. The experiments should be as transparent and unbiased as possible because a difference in these hyperparameters can affect the results significantly and cause an unfair analysis/comparison.

4.2.1 50-25-25 Approach

When the results in table 4.1 are analyzed, XLNet appears to be the strongest model across the board, achieving nearly the highest precision, recall, and F1 scores in all target classes. The statement is also satisfied when Figure 4.1 is observed as XLNet has the highest accuracy value out of all the models. Additionally, XLNet is the most consistent and balanced model amongst the others. The maximum gap between the precision and recall values is the lowest, 4% for XLNet, 17% for DistilBERT, and 7% for CNN. XLNet's superior overall performance can be attributed to its permutation-based training and use of a bidirectional context which enables it to understand the full context of each word in the input sequence. This comprehensive understanding of language allows XLNet to more accurately predict the correct class, as seen by its high precision and recall. Additionally, XLNet's sophisticated architecture is particularly effective at managing long sequences and complex language structures, which may be common in app reviews where context is key. However, the main disadvantage of XLNet was the intense runtime. The computational cost of XLNet, both runtime and CPU/GPU usage, was extraordinary. It took 20 minutes to

run a single epoch, whereas it took 10 minutes for DistilBERT and 5 seconds for CNN.

For this experiment, DistilBERT performed the worst among the models. The accuracy score is the lowest by 83% and although the average F1-score of DistilBERT is quite similar to CNN, the gap (inconsistency) between precision and recall values is the largest. For example, if the results for “bug report” are observed, it can be seen that DistilBERT has a higher F1 score than CNN. However, the precision score is 77% and the recall score is 94%, this level of inconsistency is not optimal for this task. The high recall but lower precision for DistilBERT might suggest that it is very good at identifying relevant instances (true positives), but it also misclassifies more irrelevant instances as relevant (false positives). This can occur when the model overgeneralizes from the training data, possibly due to the reduced complexity of the distilled model which may not capture the nuances as effectively as the full BERT model. DistilBERT, despite having the lowest overall accuracy, performs well in the recall metric, particularly for 'Bug Report' classification. This suggests that while DistilBERT may produce more false positives, it is less likely to miss actual bug reports, which could be highly valuable in a practical setting where identifying every potential bug is critical.

Lastly, CNN had the second-best performance among the 3 models. CNN outperformed DistilBERT and nearly matched XLNet in terms of accuracy with a score of 84.2%. This is very impressive as CNN is not a large language model that was pre-trained with a large corpus by a global company. CNN, while not reaching the same peak as XLNet, provides a respectable balance between precision and recall, as well as nearly having the same accuracy score. This shocking performance of CNN compared to the other LLM models can be explained in many ways. CNNs are particularly adept at extracting local and positionally relevant features. They can capture essential linguistic patterns within text, such as n-grams, which are crucial in identifying the sentiment and intent behind user reviews. CNNs, with their architecture and often a smaller number of trainable parameters compared to LLMs, can be more robust to overfitting, particularly when the dataset size is not massive. This can result in better generalization of the test data. The configuration of the CNN, including the choice of hyperparameters like the number of filters, size of the convolutional kernels, and the dropout rate, might have been well-optimized for the task, which could lead to it performing better than a suboptimally tuned DistilBERT. However, aside from overall performance, the most important comparison of CNN and other LLMs are the computational cost and running time. During this experiment, CNN was 240 times faster than XLNet and 120 times faster than DistilBERT in runtime. This type of difference is incredible for training models and it will certainly have a critical role in choosing the best model.

In conclusion, DistilBERT had the worst overall performance with the lowest accuracy score and the most inconsistent precision, recall scores. XLNet was the best performing model in terms of accuracy score and consistency, but the computational cost and total runtime of XLNet were the worst by far. On the other hand, despite not being a large language model, CNNs performance and consistency were nearly the

same as XLNet. On top of that, CNN was over 200 times faster, and computational cost was significantly lower. Considering all factors, I believe CNN was the best model in terms of accuracy and efficiency.

4.2.2 K-Fold Cross Validation Approach

When the results in Table 4.2 are analyzed, a significant increase in improvement can be observed in the overall performance of all 3 models. Most of the metric values increased nearly by 10%, while some of the values increased up to 17%. For example, the recall value for feature request increased from 80% to 92% in CNN, the precision value for bug report increased from 77% to 94% for DistilBERT, and the precision value for rating increased from 81% to 96% for XLNet. The improvement observed can be attributed to the use of K-fold cross-validation, which enables the model to be trained on a larger dataset by iteratively utilizing different portions of the dataset for training and testing. This approach decreases the risk of overfitting the model to the training data and enhances its generalization ability. Furthermore, cross-validation aids in identifying potential problems with the dataset and model architecture, which can be resolved to further enhance the model's performance.

Moreover, after analyzing Figure 4.2, it can be seen that DistilBERT is the best performing model in terms of accuracy with a score of 92.5%. CNN is nearly as good as DistilBERT with 92.2% and surprisingly, XLNet is the worst performing model with 90% accuracy. Just like the evaluation metrics in table 2, there is an improvement in overall performances for all 3 models. Accuracy values of both CNN and DistilBERT increased nearly by 10% and XLNet by 6%.

CNN shows remarkably consistent performance across all categories, with very little variation between precision, recall, and F1 scores. This indicates that CNN is reliably capturing features that are equally representative for all classes, a sign of a well-balanced model. CNN's relatively stable performance between experiments highlights its resilience and suggests it may be a robust choice for datasets with varying characteristics, possibly due to its ability to capture essential local features without overcomplicating the model. The total runtime was once again the lowest by far. CNN was 130 times faster than XLNet and 50 times faster than DistilBERT.

DistilBERT displays a significant improvement in recall, precision and recall for all classes, suggesting its ability to correctly identify most of the relevant instances across folds.
The highly increased F1 scores suggest that the model has improved generalization across different subsets of the data. The model's architecture, which aims to retain most of the language understanding

capabilities of BERT with fewer parameters, may have benefitted from the regularization effect that cross-validation provides. In this case, its performance suggests the model can generalize well across different subsets of the data. DistilBERT's improvement in the K-Fold experiment could mean that the architecture may have underperformed in a single test set due to overfitting or not capturing the full data complexity, and is more robust across multiple subsets. Surprisingly, XLNet, which performed best in the initial experiment now is the worst performing model in the second experiment. XLNet's performance dip in the second experiment could suggest that it was overfitted to the particular split of the first experiment, whereas the K-Fold Cross Validation revealed vulnerabilities when exposed to different subsets of data. Despite the performance drop, XLNet has the best precision on the entire table with a precision score of 96% on rating. However, considering every factor, I believe CNN is the model to choose out of the 3 because of consistency, impressive results, and fast runtime in both experiments.

The variance in results between the initial split approach and K-Fold Cross Validation could be due to how the models handle data diversity. K-Fold Cross Validation provides a more rigorous test of model generalizability by training and evaluating multiple data subsets, which can highlight issues like overfitting that might not be apparent in a single train-test split.

In conclusion, the K-Fold Cross Validation results emphasize the importance of using rigorous validation techniques to understand model performance fully. While XLNet excelled in the split approach, it was less consistent across multiple folds, indicating that the best-performing model in one scenario may not always hold its lead in others, particularly when the validation method changes. DistilBERT's stronger showing in K-Fold Cross Validation highlights its potential as a more generalizable model in this context, and CNN's steady performance solidifies its status as the most reliable and robust classifier for text data. The differences between the two sets of results highlight the complexity of NLP tasks and the importance of model selection based on the specific characteristics of the dataset and the desired balance between precision and recall. They also underscore the necessity of rigorous model validation methods to ensure that models will perform well in practice, not just on a particular partition of the data.

5. Research Evaluation

The section reflects on the strengths and limitations of the study and compares it to other related works.

5.1 Comparison with Related Work

This section evaluates the project's performance by analyzing it alongside similar studies. This comparison highlights the project's grasp of existing academic discussions and pinpoints the unique contributions it makes.

In comparison to the “Ensemble Methods for App Review Classification: An Approach for Software Evolution” paper, Guzman et al. reported that the machine learning methods such as Naive Bayes, SVM and Logistic Regression, generally performed better than individual classifiers, achieving an average precision of 0.74 and a recall of 0.59 [43]. The best performing method resulted in an F1 score of around 0.64. Our project managed the outperform this study by a significant amount as the best F1 score for experiment one was 0.88 and 0.95 for the second experiment.

When compared with the “On the automatic classification of app reviews” paper, it can be observed that our study managed to outperform most of the models in the study. Maalej et al. experimented with different classification models to classify reviews into feature request, bug report, rating, and user experience [42]. Most of the model's precision, recall and f1 score values are below 0.80 while in our experiments, nearly every value is between 0.80-0.95.

The usage of deep learning/large language models like CNN, DistilBERT, and XLNet brings a significant advantage in terms of contextual understanding and semantic processing. Also, factors like hyperparameter tuning, balanced datasets and dataset sizes can have a critical impact on the outperforming.

The research introduces a relative analysis of state-of-the-art LLMs with a traditional CNN model, using a detailed methodological approach. The inclusion of K-Fold Cross Validation represents a comprehensive evaluation of model stability and generalization. The results indicate that while traditional CNNs can achieve high performance, LLMs like DistilBERT and XLNet may offer enhanced capabilities, especially in complex or context-heavy classification tasks. The advanced LLMs seem to perform differently under various evaluation approaches (initial split versus K-Fold), which provides deeper insights into how such models can be fine-tuned and optimized for specific use cases in the domain of app review analysis.

In summary, this study contributes to the field by bringing state-of-the-art deep learning approaches to the challenge of classifying app reviews, showcasing a progression from traditional machine learning

methods to sophisticated models that can capture complex language patterns for a nuanced understanding of user feedback. This work underscores the ongoing shift towards leveraging more advanced AI for requirements engineering in software development.

5.2 Validity Threats

In this section, the potential limitations and validity threats encountered during the execution of the project are explored.

1- Small Dataset Size: The dataset containing 8000 balanced reviews is relatively small for deep learning applications, even though it was enough for initial model training and validation. Deep learning models, particularly sophisticated architectures like XLNet and DistilBERT, typically require large amounts of data to fully capture the complexities of language and improve generalization. There is a potential that with a larger and more varied dataset, the performance of these models could increase, as they would have more examples from which to learn and better opportunities to fine-tune their parameters.

2- Sampling Bias: The dataset for training and validating the models included app reviews, which may not reflect user feedback from different platforms or app categories. This could potentially limit the generalizability of the models to other contexts or types of apps not covered in the dataset.

3- Model Overfitting: There is always a risk that complex models like XLNet and DistilBERT could overfit the training data, especially given their capacity to model extensive contextual information. Overfitting would impair the models' ability to generalize to new, unseen data.

4- High Runtime and Computational Cost: During the fine-tuning and testing of DistilBERT and XLNet, the runtime of these models was significantly higher than CNN. For example, during the K-Fold Cross Validation, the total runtime of XLNet was 8.5 hours and DistilBERT’s was 4 hours. The amount of time required to complete the experiments was a huge limitation.

5- Evaluation Metrics Limitations: The primary metrics used to evaluate the models were precision, recall, and F1-score. While these metrics provide a comprehensive view of model performance, they might not capture all aspects of model effectiveness, such as the ability to handle nuanced

linguistic features or the models' performance on particularly challenging or subtle categories of reviews. Including additional metrics like AUC-ROC or confusion matrices could provide deeper insights into model performance across various thresholds and decision boundaries.

By recognizing these threats and addressing them, the study becomes more credible and lays down a solid foundation for future research. It also makes other researchers aware of the possible limitations, helping them to take these into account when they use or expand on the findings.

5.3 Summary of Key Research Findings

The study has provided several significant findings, contributing valuable insights to the field of natural language processing and software development. Here are the key findings from this study:

Effectiveness of Advanced Language Models: The study demonstrated that advanced language models such as XLNet, DistilBERT, and CNN are highly effective in classifying app reviews into categories such as bug reports, feature requests, user experiences, and ratings. These models not only performed well in terms of precision and recall but also showed robustness across different validation methods including K-Fold Cross Validation, which enhances the credibility of their performance metrics.

Variability in Model Performance: The research highlighted variability in model performance across different evaluation setups. For instance, while XLNet performed exceptionally well in the initial data split, it was the least effective during the K-Fold Cross Validation, suggesting sensitivity to the training and validation data setup. This contrast underscores the importance of rigorous model evaluation and selection depending on specific use cases.

Generalizability and Robustness of CNN: The CNN model, while simpler than the transformer-based models like XLNet and DistilBERT, showed impressive generalizability and robustness across different datasets and validation scenarios. This suggests that CNNs remain a valuable tool for text classification tasks, particularly when computational resources or data availability are limited.

The successful classification of app reviews using advanced deep learning models like CNN, DistilBERT, and XLNet holds significant value for the field of software development. By accurately

categorizing user feedback into distinct aspects such as bug reports, feature requests, and user experiences, these models enable developers and product managers to efficiently address user needs and prioritize development efforts.

6. Conclusion

6.1 Achievements and Contribution

The aim of this project was to analyze and compare the performance of 3 different advanced models on a multiclass text classification task. The task is the classification of user app reviews into 4 different categories which are bug report, feature request, user experience and rating. Through detailed analysis and extensive validation, including both initial splits and K-Fold Cross Validation, the project highlighted the strengths and weaknesses of each model, offering critical insights into their suitability for handling complex natural language processing tasks.

The CNN model demonstrated remarkable robustness and generalizability, performing consistently well across different validation setups. While it is less complex than transformer-based models, its ability to capture essential textual features without extensive computational resources makes it a valuable tool for practical applications. DistilBERT was the best performing model in K-Fold Cross Validation, indicating its effectiveness in generalizing across different data subsets. This adaptability makes it especially useful for real-world applications where data variability is common. XLNet showed impressive results in the initial data split with its ability to understand complex linguistic structures. However, during K-Fold Cross Validation, XLNet's performance was less consistent, suggesting sensitivity to variations in data subsets and training setups.

The project successfully met its objective of collecting and preprocessing diverse datasets that consisted of clean, relevant, and consistent user reviews. The size of the datasets was not as big as preferred but was certainly enough for this project. All the preprocessing tasks were executed accurately. The CNN, DistilBERT, and XLNet models were designed to suit multiclass text classification with an appropriate level of computational complexity. Both primary experiments, the 50-25-25 split and K-Fold Cross Validation, were successfully conducted with various hyperparameters. Results from the experiments were analyzed in detail using a thorough set of evaluation metrics. All important results and settings used in the experiments were carefully recorded for future analysis. Statistical tools and visual aids like tables and charts were used effectively to illustrate how different models performed, offering clear insights into the best model configurations for specific scenarios.

Through this project, I gained a deep understanding of the capabilities and limitations of each model used—CNN, DistilBERT, and XLNet—in handling complex linguistic data. I explored in depth how each model processes and classifies text, revealing distinct strengths and weaknesses. For instance, I learned that CNNs, while less sophisticated than models like XLNet, offer robust performance and generalizability with less computational demand. DistilBERT, being a lighter version of the more robust BERT model, provided a balance between performance and efficiency, making it highly suitable for environments with limited computational resources. XLNet's ability to handle context more effectively stood out, although its sensitivity to training configurations posed challenges in consistency across different tests.

The hands-on experience I gained from managing datasets, fine-tuning the parameters of complex models, and interpreting the outcomes significantly enhanced my proficiency in natural language processing tasks. This practical involvement helped me understand not just the theoretical aspects of deep learning, but also the decision-making needed to optimize performance across various scenarios. Additionally, this project taught me the value of a systematic approach to data preprocessing, choosing the right models, and analyzing outcomes. I learned the importance of carefully documenting every step to ensure accuracy and consistency, which has equipped me for future projects and practical applications. These skills are crucial as they go beyond theoretical knowledge, providing practical insights that are directly applicable to real-world industry challenges.

However, every project comes with limitations. For this project, I believe the dataset is relatively small. Even though it was enough to complete this project and achieve impressive results, I think a bigger dataset would increase the performance of all 3 models. Furthermore, models like XLNet demonstrated variability in performance across different training and validation scenarios, indicating potential issues with overfitting and adaptability to different data distributions. The computational requirements also presented a significant challenge, as advanced models like DistilBERT and XLNet need substantial resources for training and validation, which could restrict their application in environments with limited resources. These challenges highlight important areas for future research, including expanding the dataset, refining training methods, and picking models with less computational cost.

6.2 Future Work

Based on the insights and limitations highlighted by this project, numerous directions for future research have been identified, each focused on improving the effectiveness and practicality of deep learning models in classifying app reviews and related tasks.

Dataset Enhancement: An essential next step is to expand the collection of datasets to include a greater

variety of reviews. This will improve model accuracy by training on a broader spectrum of user feedback, which includes more expressions and contexts.

Preventing Model Overfitting: The risk of overfitting, particularly with complex models like XLNet and DistilBERT, points towards the necessity for improved regularization techniques and model validation strategies. Future work could explore the development of advanced training methodologies that better balance model complexity with the ability to generalize to new data.

Multi-label Classification Development: Some reviews can talk about multiple aspects of the application. So, there is a need to develop frameworks that can handle reviews mentioning multiple product aspects. Assigning multiple labels to such reviews would provide a more accurate description of their content.

Reducing Runtime and Computational Costs: The high runtime and computational costs associated with models like DistilBERT and XLNet highlight the importance of optimizing these models for efficiency. Research could focus on refining these models through techniques such as model pruning or the development of more efficient architectures that reduce computational demands without compromising performance.

Hybrid Model Approaches: Combining the predictive power of different models through ensemble methods or hybrid architectures. This approach could lead to better classification performance combining the strengths of multiple models.

Enhancing Evaluation Metrics: The limitations of current evaluation metrics like precision, recall, and F1-score indicate a need for incorporating additional metrics such as AUC-ROC or confusion matrices in future studies. These metrics could provide a deeper understanding of model performance, especially in handling subtle linguistic features and complex review categories.

These directions not only aim to address the specific challenges faced in this project but also to push the boundaries of what can be achieved with AI and machine learning in processing complex textual data, thereby enhancing their practical applications across various sectors.

References:

[1] H. Wu, W. Deng, X. Niu and C. Nie, "Identifying Key Features from App User Reviews," 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), Madrid, ES, 2021, pp. 922-932, doi: 10.1109/ICSE43902.2021.00088.

[2] M. Ben Messaoud, Ilyes Jenhani, Nermine Ben Jemaa and Mohamed Wiem Mkaouer (2019). A Multi-label Active Learning Approach for Mobile App User Review Classification. pp.805–816. doi:https://doi.org/10.1007/978-3-030-29551-6_71.

[3] E. Noei, F. Zhang and Y. Zou, "Too Many User-Reviews! What Should App Developers Look at First?," in *IEEE Transactions on Software Engineering*, vol. 47, no. 2, pp. 367-378, 1 Feb. 2021, doi: 10.1109/TSE.2019.2893171.

[4] D. Pagano and W. Maalej, "User feedback in the appstore: An empirical study," 2013 21st IEEE International Requirements Engineering Conference (RE), Rio de Janeiro, Brazil, 2013, pp. 125-134, doi: 10.1109/RE.2013.6636712.

[5] W. Maalej and H. Nabil, "Bug report, feature request, or simply praise? On automatically classifying app reviews," 2015 IEEE 23rd International Requirements Engineering Conference (RE), Ottawa, ON, Canada, 2015, pp. 116-125, doi: 10.1109/RE.2015.7320414.

[6] Khalid, M., Shehzaib, U. and Asif, M. (2015). A Case of Mobile App Reviews as a Crowdsource. International Journal of Information Engineering and Electronic Business, 7(5), pp.39–47. doi:https://doi.org/10.5815/ijieeb.2015.05.06.

[7] Khalid, Mubasher, Usman Shehzaib, and Muhammad Asif. "A case of mobile app reviews as a crowdsource." International Journal of Information Engineering and Electronic Business 7.5 (2015): 39.

[8] Apple (2019). Ratings, Reviews, and Responses - App Store - Apple Developer. Apple.com. Available at: https://developer.apple.com/app-store/ratings-and-reviews/.

[9] Panichella, S., Di Sorbo, A., Guzman, E., Visaggio, C.A., Canfora, G. and Gall, H.C. (2015). How can i improve my app? Classifying user reviews for software maintenance and evolution. IEEE Xplore. doi:https://doi.org/10.1109/ICSM.2015.7332474.

[10] Suthaharan, Shan. "Machine learning models and algorithms for big data classification." Integr. Ser. Inf. Syst 36 (2016): 1-12.

[11] Song, C., Ristenpart, T. and Shmatikov, V. (2017). Machine Learning Models that Remember Too Much. Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. doi:https://doi.org/10.1145/3133956.3134077.

[12] El Naqa, I. and Murphy, M.J. (2015). What Is Machine Learning? Machine Learning in Radiation

Oncology, 1(1), pp.3–11. doi:https://doi.org/10.1007/978-3-319-18305-3_1.

[13] Sarker, I.H. (2021). Machine Learning: Algorithms, Real-World Applications and Research Directions. SN Computer Science, 2(3), pp.1–21. doi:https://doi.org/10.1007/s42979-021-00592-x.

[14] Suthaharan, S. (2016). Supervised Learning Algorithms. Integrated series on information systems, pp.183–206. doi:https://doi.org/10.1007/978-1-4899-7641-3_8.

[15] Cunningham, P., Cord, M. and Delany, S.J. (2008). Supervised Learning. Machine Learning Techniques for Multimedia, pp.21–49. doi:https://doi.org/10.1007/978-3-540-75171-7_2.

[16] Hastie, T., Tibshirani, R. and Friedman, J. (2008). Overview of Supervised Learning. The Elements of Statistical Learning, pp.1–33. doi:https://doi.org/10.1007/b94608_2.

[17] Jansson, P.A. (1991). Neural Networks: An Overview. Analytical Chemistry, 63(6), pp.357A362A. doi:https://doi.org/10.1021/ac00006a739.

[18] Picton, P. (1994). What is a Neural Network? Introduction to Neural Networks, pp.1–12. doi:https://doi.org/10.1007/978-1-349-13530-1_1.

[19] Krogh, A. What are artificial neural networks?. Nat Biotechnol 26, 195–197 (2008). https://doi.org/10.1038/nbt1386

[20] Buckner, C. (2019). Deep learning: A philosophical introduction. Philosophy Compass, 14(10). doi:https://doi.org/10.1111/phc3.12625.

[21] P. P. Shinde and S. Shah, "A Review of Machine Learning and Deep Learning Applications," 2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA), Pune, India, 2018, pp. 1-6, doi: 10.1109/ICCUBEA.2018.8697857.t

[22] Caroppo, A., Leone, A. and Siciliano, P. (2020). Comparison Between Deep Learning Models and Traditional Machine Learning Approaches for Facial Expression Recognition in Ageing Adults. Journal of Computer Science and Technology, 35(5), pp.1127–1146. doi:https://doi.org/10.1007/s11390-020-9665-4.

[23] S. Khan, M. Sajjad, T. Hussain, A. Ullah and A. S. Imran, "A Review on Traditional Machine Learning and Deep Learning Models for WBCs Classification in Blood Smear Images," in IEEE Access, vol. 9, pp. 10657-10673, 2021, doi: 10.1109/ACCESS.2020.3048172.

[24] Li, Q., Peng, H., Li, J., Xia, C., Yang, R., Sun, L., Yu, P.S. and He, L. (2022). A Survey on Text

Classification: From Traditional to Deep Learning. ACM Transactions on Intelligent Systems and Technology, 13(2), pp.1–41. doi:https://doi.org/10.1145/3495162.

[25] Albawi, S., Mohammed, T.A. and Al-Zawi, S. (2017). Understanding of a Convolutional Neural Network. 2017 International Conference on Engineering and Technology (ICET), pp.1–6. doi:https://doi.org/10.1109/icengtechnol.2017.8308186.

[26] Z. Li, F. Liu, W. Yang, S. Peng and J. Zhou, "A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects," in IEEE Transactions on Neural Networks and Learning Systems, vol. 33, no. 12, pp. 6999-7019, Dec. 2022, doi: 10.1109/TNNLS.2021.3084827.

[27] P. Song, C. Geng and Z. Li, "Research on Text Classification Based on Convolutional Neural Network," 2019 International Conference on Computer Network, Electronic and Automation (ICCNEA), Xi'an, China, 2019, pp. 229-232, doi: 10.1109/ICCNEA.2019.00052.

[28] Yamashita, R., Nishio, M., Do, R.K.G. et al. Convolutional neural networks: an overview and application in radiology. Insights Imaging 9, 611–629 (2018). https://doi.org/10.1007/s13244-018-0639-9

[29] Kasneci, E., Sessler, K., Küchemann, S., Bannert, M., Dementieva, D., Fischer, F., Gasser, U., Groh, G., Günnemann, S., Hüllermeier, E., Krusche, S., Kutyniok, G., Michaeli, T., Nerdel, C., Pfeffer, J., Poquet, O., Sailer, M., Schmidt, A., Seidel, T. and Stadler, M. (2023). ChatGPT for good? On opportunities and challenges of large language models for education. Learning and Individual Differences, 103, p.102274. doi:https://doi.org/10.1016/j.lindif.2023.102274.

[30] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A., Kaiser, Ł. and Polosukhin, I. (2017). Attention Is All You Need. Available at: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[31] Strasser, Anna. "On pitfalls (and advantages) of sophisticated large language models." arXiv preprint arXiv:2303.17511 (2023).

[32] Casola, S., Lauriola, I. and Lavelli, A. (2022). Pre-trained transformers: an empirical comparison. Machine Learning with Applications, 9, p.100334. doi:https://doi.org/10.1016/j.mlwa.2022.100334.

[33] Sanh, V., Debut, L., Chaumond, J. and Wolf, T. (2020). DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. Available at: https://arxiv.org/pdf/1910.01108.pdf.

[34] Dogra, V., Singh, A., Verma, S., Kavita, Jhanjhi, N.Z., Talib, M.N. (2021). Analyzing DistilBERT for

Sentiment Classification of Banking Financial News. In: Peng, SL., Hsieh, SY., Gopalakrishnan, S., Duraisamy, B. (eds) Intelligent Computing and Innovation on Data Science. Lecture Notes in Networks and Systems, vol 248. Springer, Singapore. https://doi.org/10.1007/978-981-16-3153-5_53

[35] Yang, Zhilin, et al. "Xlnet: Generalized autoregressive pretraining for language understanding." Advances in neural information processing systems 32 (2019).

[36] Yan, R., Jiang, X. & Dang, D. Named Entity Recognition by Using XLNet-BiLSTM-CRF. Neural Process Lett 53, 3339–3356 (2021). https://doi.org/10.1007/s11063-021-10547-1

[37] A. F. Adoma, N. -M. Henry and W. Chen, "Comparative Analyses of Bert, Roberta, Distilbert, and Xlnet for Text-Based Emotion Recognition," 2020 17th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP), Chengdu, China, 2020, pp. 117-121, doi: 10.1109/ICCWAMTIP51612.2020.9317379.

[38] T. -T. Wong and P. -Y. Yeh, "Reliable Accuracy Estimates from k-Fold Cross Validation," in IEEE Transactions on Knowledge and Data Engineering, vol. 32, no. 8, pp. 1586-1594, 1 Aug. 2020, doi: 10.1109/TKDE.2019.2912815.

[39] Fushiki, T. Estimation of prediction error by using K-fold cross-validation. Stat Comput 21, 137–146 (2011). https://doi.org/10.1007/s11222-009-9153-8

[40] Anguita, D., Ghelardoni, L., Ghio, A., Oneto, L. and Ridella, S. (2012). The 'K' in K-fold Cross Validation. Available at: https://www.esann.org/sites/default/files/proceedings/legacy/es2012-62.pdf.

[41] Jung, Y. (2018). Multiple predicting K-fold cross-validation for model selection. Journal of Nonparametric Statistics, 30(1), 197–215. https://doi.org/10.1080/10485252.2017.1404598

[42] Maalej, W., Kurtanović, Z., Nabil, H. and Stanik, C. (2016). On the automatic classification of app reviews. Requirements Engineering, 21(3), pp.311–331. doi:https://doi.org/10.1007/s00766-016-0251-9.

[43] E. Guzman, M. El-Haliby and B. Bruegge, "Ensemble Methods for App Review Classification: An Approach for Software Evolution (N)," 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), Lincoln, NE, USA, 2015, pp. 771-776, doi: 10.1109/ASE.2015.88.

[44] Emmanuel, D. (2023). drmingler/Automatic-App-Reviews-Classification-Model. GitHub. Available at: https://github.com/drmingler/Automatic-App-Reviews-Classification-Model [Accessed 15 Apr. 2024].

[45] Yahui Chen (2015). Convolutional Neural Network for Sentence Classification.

UWSpace. http://hdl.handle.net/10012/9592

[46] Goutte, C., Gaussier, E. (2005). A Probabilistic Interpretation of Precision, Recall and F-Score, with Implication for Evaluation. In: Losada, D.E., Fernández-Luna, J.M. (eds) Advances in Information Retrieval. ECIR 2005. Lecture Notes in Computer Science, vol 3408. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-31865-1_25

[47] Yacouby, R. and Axman, D. (2020). Probabilistic Extension of Precision, Recall, and F1 Score for More Thorough Evaluation of Classification Models. ACLWeb. doi:https://doi.org/10.18653/v1/2020.eval4nlp-1.9.

Figure 2.1: www.superannotate.com. (2023). What is image classification? Basics you need to know | SuperAnnotate. Available at: https://www.superannotate.com/blog/image-classification-basics.

Figure 2.2: Lavanya Shukla (2019). Designing Your Neural Networks. Medium. Available at: https://towardsdatascience.com/designing-your-neural-networks-a5e4617027ed.

Figure 2.3: Alsaleh, A. and Perkgoz, C. (2023). A space and time efficient convolutional neural network for age group estimation from facial images. PeerJ Computer Science, 9, p.e1395. doi:https://doi.org/10.7717/peerj-cs.1395.

Figure 2.4: Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A., Kaiser, Ł. and Polosukhin, I. (2017). Attention Is All You Need. Available at: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Figure 2.5: Adel, Hadeer & Dahou, Abdelghani & Mabrouk, Alhassan & Elsayed Abd Elaziz, Mohamed & Kayed, Mohammed & El-henawy, Ibrahim & Alshathri, Samah & Ali, Abdelmgeid. (2022). Improving Crisis Events Detection Using DistilBERT with Hunger Games Search Algorithm. Mathematics. 10. 447. 10.3390/math10030447.

Figure 2.6: Ray, T. (2019). Google's latest language machine puts emphasis back on language. ZDNET. Available at: https://www.zdnet.com/article/googles-latest-language-machine-puts-emphasis-back-on-language/

Figure 3.3: Kosar, V. (2022). Tokenization in Machine Learning Explained. vaclavkosar.com. Available at: https://vaclavkosar.com/ml/Tokenization-in-Machine-Learning-Explained.