JOHANNES KEPLER
UNIVERSITY LINZ

Submitted by
Philipp Engelbrechtsmüller

Submitted at
Institute for Business Informatics - Data & Knowledge Engineering

Supervisor
o. Univ.-Prof. Dipl.-Ing. Dr. techn. Michael Schrefl

Co-Supervisor
Martin Straßer, MSc

June 2024

Fine-Tuning Large Language Models for Ticket Classification at Doka GmbH

Master Thesis
to obtain the academic degree of
Master of Science
in the Master's Program
Business Informatics

JOHANNES KEPLER UNIVERSITY LINZ
Altenbergerstraße 69
4040 Linz, Österreich
www.jku.at

Kurzfassung

Mit der fortschreitenden Digitalisierung nimmt die Zahl der IT-Anwendungen in Unternehmen stetig zu. Dies führt zu einer Zunahme von IT-Problemen, die in großen Unternehmen meist über ein IT-Ticketsystem abgewickelt werden. Auch die Doka GmbH nutzt einen solchen Helpdesk, um Tickets manuell an die entsprechende Abteilung zur Bearbeitung weiterzuleiten. Dieser manuelle Prozess ist sehr zeit- und kostenaufwändig. Um diesen Prozess zu automatisieren, ist eine automatische Textklassifizierung erforderlich. In der Vergangenheit wurde die Textklassifizierung mit traditionellen Klassifikatoren und später mit Deep Neural Networks (DNN) durchgeführt. Mit der Einführung der Transformer-Architektur wurden große Sprachmodelle (LLM) wie GPT und BERT entwickelt. GPT ist ein von OpenAI veröffentlichtes LLM, während BERT von Google entwickelt wurde. Diese Modelle sind sehr gut im Verstehen von allgemeinem Text, versagen aber oft beim Verstehen von domänenspezifischen Wörtern, wie sie in IT-Tickets vorkommen. Aus diesem Grund werden fein abgestimmte Modelle von GPT und BERT für die fünf Attribute, die ein Ticket bei der Doka GmbH beschreiben, erstellt und verglichen. Bei der Feinabstimmung wird das LLM auf eine bestimmte Aufgabe trainiert und die Modellparameter werden während des Trainings an diese Aufgabe angepasst. Es stellt sich heraus, dass GPT das BERT-Modell in 4 von 5 Fällen in Bezug auf die Genauigkeit übertrifft. Um eine bessere Leistung mit dem BERT-Modell zu erreichen, werden Data Augmentation Techniken wie EDA (Easier Data Augmentation) und AEDA (An Easier Data Augmentation) verwendet. Diese verwenden Methoden wie die Random Insertion, um sicherzustellen, dass zusätzliche klassifizierte Daten generiert werden können. Die Untersuchung zeigt, dass beide Techniken in der Lage sind, die Leistung des BERT-Modells erheblich zu verbessern.

Abstract

As digitalization progresses, the number of IT applications in companies constantly increases. This leads to an increase in IT problems, which are usually handled by an IT ticket system in large companies. Doka GmbH also uses such a helpdesk to manually forward tickets to the responsible department for processing. This manual process is very time-consuming and costly. Automatic text classification is required to automate this process. In the past, text classification was performed using traditional classifiers and, later, deep neural networks (DNN). With the advent of the Transformer architecture, Large Language Models (LLM) such as GPT and BERT were developed. GPT is an LLM published by OpenAI, while Google developed BERT. These are very good at understanding general text but often fail to understand domain-specific text such as that found in IT tickets. For this reason, fine-tuned models of GPT and BERT are created and compared for the five attributes that describe a ticket at Doka GmbH. Fine-tuning involves training the LLM on a specific task and adjusting the model parameters to suit that task during training. It turns out that GPT outperforms BERT in terms of accuracy in 4 out of 5 cases. Data augmentation techniques such as EDA (Easier Data Augmentation) and AEDA (An Easier Data Augmentation) are used to achieve better performance with the fine-tuned BERT model. These use methods such as random insertion to ensure that additional labeled data can be generated. The research shows that both techniques can improve the performance of the BERT model.

Contents

List of Figures

List of Tables

Listings

List of Abbreviations

NLP Natural Language Processing

RNN Recurrent Neural Networks

LSTM Long Short-Term Memory Networks

CNN Convolutional Neural Networks

BERT Bidirectional Encoder Reprepresentations from Transformers

GPT Generative Pre-Trained Transformer

DA Data Augmentation Techniques

EDA Easy Data Augmentation Techniques

AEDA An Easier Data Augmentation Technique

TF-IDF Term Frequency-Inverse Document Frequence

LLM Large Language Models

ELMo Embeddings from Language Models

MLM Masked Language Model

NSP Next Sentence Prediction

1 Introduction

As digitalization progresses across all areas of the business, the number of different IT applications in use is constantly increasing (Revina et al., 2020). This requires efficiently handling IT problems as part of problem management to cope with the increasing complexity. According to the IT Information Library (ITIL), Problem Management is minimizing the impact of an incident or problem. An incident is an event that is not part of standard operations and impacts service operations (ITIL, 2024). It should also be noted that improved IT service quality is essential to business growth (Diao et al., 2009). To deal with problems quickly, these incidents are usually handled through an IT ticket system. These tickets must be routed to the correct department/team with the proper prioritization (Revina et al., 2020).

Also, Doka GmbH is facing increasing complexity in IT services. Doka GmbH, headquartered in Amstetten and a subsidiary of Umdasch Group AG, is a leading international player in the formwork and scaffolding industry. Doka GmbH offers various formwork solutions for construction, civil engineering, and tunneling. These include wall and slab formwork, beam formwork, climbing formwork, and systems for exposed concrete applications. Turnover in 2022 totaled 1.788 billion euros. Over 7.000 employees work at 178 locations worldwide, across 60 countries (Doka, 2024).

With so many employees, there is a large number of inquiries and problems. New employees must be created in the relevant systems; data must be correctly maintained in the systems, and, of course, software problems must also be able to be reported. To handle this organisationally, Doka Gmbh uses a so-called Helpdesk system. In this system, employees can create a ticket directly or by e-mail, which the responsible team must process. In this way, around 80.000 tickets are made by employees every year. Essential parameters such as Priority and the correct Serviceline must be set to ensure efficient processing. Most tickets are reviewed manually by the helpdesk team and then assigned to the responsible Serviceline. The Serviceline stands for the responsible department or team.

To avoid manual allocation of IT tickets, automatic IT ticket classification has become increasingly relevant (Revina et al., 2020). The task of ticket classification is part of the document classification class in computer science (Diao et al., 2009). This, in turn, is part of text classification. Text classification is one of the core topics in Natural Language Processing (NLP) (Sun et al., 2019). Text classification has been the subject of scientific research for many years. The aim is to classify textual information into pre-defined categories (Jiao, 2023). Conventional approaches begin with a textual analysis or featurisation, in which the text is tiled into features (Yu, 2008). Methods such as stop words, stemming, and compound term processing are applied. Before the text can be classified, it is necessary to prepare the text for machine understanding using word embeddings (Q. Li et al., 2022). There are several approaches, such as Term Frequency-Inverse Document Frequency (TF-IDF) (Baeza-Yates, Ribeiro-Neto et al., 1999), word2vec (Mikolov et al., 2013) or GloVe (Pennington et al., 2014), to name just a few. Once the text is prepared, the actual classification can be done. All known classifiers such as Decision Tree, SVM, and KNN, as well as newer ones such as XGBoost, can be used for this ((Qi, 2020)(Shah et al., 2020)).

From the 2010s (Q. Li et al., 2022), it became more and more common to use deep learning methods (DNN) for this purpose. Recurrent Neural Networks (RNNs) (Sherstinsky, 2020) or Long Short-Term Memory Networks (LSTM) (Hochreiter & Schmidhuber, 1997) are used for this purpose and achieve better results in the task of text classification than conventional classifiers. Convolutional Neural Networks (CNN) (Albawi et al., 2018), initially developed for image processing, were also used for text or ticket classification.

These deep learning models have long been considered state-of-the-art, but this changed abruptly when Vaswani et al. (2017) presented the Transformer architecture. The Transformer is purely based on attention and does not use recursion or convolution. Because the model relies on its own attention, it can weigh the importance of different parts of the input data differently, resulting in more differentiated and context-aware representations. By utilizing self-attention, the Transformer can process all parts of the input data in parallel, drastically reducing training times and improving the model's ability to capture complex relationships within the data. Based on this architecture, the Bidirectional Encoder Representations from Transformers, BERT (Devlin et al., 2018) for short, and the Generative Pre-Trained Transformer, GPT (Radford et al., 2018) for short, were developed. Although both differ slightly in their architecture, both are, in principle, first pre-trained on a large text corpus. They can then be fine-tuned to a specific task or domain. Fine-tuning aims to give the pre-trained model labeled training data in order to train the parameters for a particular task and thus ensure better performance (Sun et al., 2019). Unlike conventionally pre-trained models, BERT and GPT are pre-trained on a large-scale corpus. The increasing number of parameter scales means these models improve performance and bring new capabilities, such as in-context learning. Due to their sheer size, they are subsumed under the term large language models (LLM) (W. X. Zhao et al., 2023).

There are already numerous procedures in the literature on best fine-tuning BERT for text classification (Sun et al., 2019). For example, BERT has been fine-tuned to identify fake news in tweets (Qasim et al., 2022) or to classify IT tickets (Z. Liu et al., 2023), as in this thesis. As GPT is also a zero-shot-learner or a few-shot-learner, the literature on GPT usually only contains examples of how text classification ((Mayer et al., 2023)(B. Zhao et al., 2023)) has been implemented with these approaches. These two approaches use only a tiny amount of training data. However, since the Doka GmbH ticket classification involves many different classes, this will be implemented using the fine-tuning approach. This thesis aims to fine-tune BERT and GPT and compare how well they can classify the tickets. For this purpose, a separate model will be created for each target variable, which will then be compared, especially regarding accuracy.

As is often the case with such machine learning tasks, sufficient training data is essential. Some classes are also underrepresented in this data set. To overcome this problem, it is possible to artificially generate additional training data using data augmentation (DA) methods. There are several approaches to this, such as Easy Data Augmentation Techniques (EDA) (Wei & Zou, 2019), where techniques such as random insertion, random deletion, and random swap are used, or translating the data into another language and translating it back to generate additional data (Xie et al., 2020). It is also possible to create diverse data using the An Easier Data Augmentation Technique (AEDA) (Karimi et al., 2021), where punctuation is simply randomly inserted into sentences. This thesis aims to overview the methods used to generate artificial data using EDA and AEDA. The models are also enriched with this artificial data, and it is evaluated whether it is helpful to provide a more balanced training data set in this way or whether it even has a negative effect on the performance of the models.

1.1 Research questions

Based on the presentation of the current state of research in the introduction and the identification of the research gap, two research questions and objectives were developed, which are to be answered in this master thesis:

RQ1: How does a fine-tuned BERT model perform in the text classification task of Doka GmbH Helpdesk tickets compared to a fine-tuned GPT-3.5 model?

To answer this question, the fine-tuned BERT models are compared to fine-tuned GPT models. Doka GmbH provided five data sets to predict the target variables: Serviceline, Servicetype, Priority, Queuename, and Form. A separate model is trained for each of these target variables. The comparison between the two models is based on accuracy. The data sets vary in size, containing around 107.000 training examples. The main contribution of this thesis is creating a working BERT model that can predict the correct target variable with high accuracy. The GPT model has already been realized in mutual exchange, mainly by Doka GmbH.

RQ2: How do data augmentation strategies impact the performance of a fine-tuned BERT model in predicting the target variables "Serviceline" and "Servicetype"?

The second research question aims to answer whether data augmentation strategies can positively influence the performance of fine-tuned BERT models. Additional training data is artificially generated for the Servicetype and Serviceline models to answer this question. The EDA approach combines four methods. The Random Insertion, Random Swap, Random Deletion, and Synonym Replacement approaches are combined. In comparison, the AEDA approach will also be implemented. In this case, the additional training data generation should help compensate for the unbalanced data set. Accuracy is again used as a metric to enable a comparison.

1.2 Thesis structure

This master's thesis is organized as follows: In the second chapter, already established methods such as traditional classifiers, CNNS, RNNs, and LSTMs are presented, and how they tackle the text classification task. The third chapter focuses on the transformer approach, which forms the basis for developing models such as BERT and GPT. These models represent the state of the art in automatic text processing and offer new perspectives for text classification. The fourth chapter shows how these models have been used in practical applications to improve the efficiency and accuracy of ticket classification. In the fifth chapter, the work shifts to presenting the dataset used and conducting a detailed data exploration. The sixth chapter shows training the two models, BERT and GPT. A comprehensive evaluation of the model performance is provided in the seventh chapter. Here, the results of the applied models are analyzed and evaluated. In the eighth chapter, the work is extended by integrating data augmentation techniques, particularly the EDA and the AEDA approach, to improve model performance further. It analyzes how the accuracy of the models in text classification can be increased through targeted changes and extensions to the training data set. The paper concludes with a summary of the findings, a discussion of the limitations, and an outlook on future research directions.

2

Approaches for Text Classification

In this section of the paper, we will take a closer look at the field of text classification. This area has been researched for a long time and is becoming increasingly important. Over time, different approaches and methods have been used. These have been replaced by other methods as technology has progressed. The categorization roughly follows that of Li et al. (Q. Li et al., 2022), distinguishing between traditional and deep learning methods. Fig. 2.1 shows the text classification process in detail. The text to be processed serves as the basis. This can be documents, emails, or helpdesk tickets, as in this thesis. During preprocessing, the text is prepared by removing stopwords, lowercasing, and using methods such as stemming. With traditional methods, an intermediate step, feature extraction, is necessary first. Here, the text is converted into numerical representations. Methods such as Bag of Words or TF-IDF are used here, which will be explained in more detail later. After this step, classifiers such as Naive Bayes, Decision Tree, or Random Forest can be used. In contrast, deep learning models do not require manual feature extraction. They automatically learn the features from the text. After the models have been trained, they are evaluated using metrics such as accuracy or F1 score, and texts can be classified using the trained models.

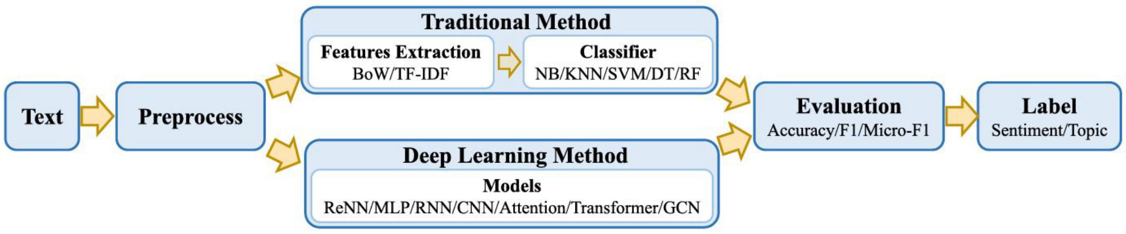


Figure 2.1: Text classification methods (Q. Li et al., 2022)

The focus is on concrete implementations of traditional and deep learning methods for text classification. Of course, one should not forget the rule-based approaches already used for ticket classification (Diao et al., 2009). There are also examples (Hadi et al., 2018) where rule-based and classifier-based approaches have been combined and achieved excellent results. However, these will not be further discussed in detail in this paper.

Doka GmbH has been trying to automate ticket classification for some time. For this reason, attempts have already been made to implement such a classification using traditional methods. However, the results fell short of expectations. This chapter and chapters three and four aim to show how LLM works compared to traditional approaches and why it delivers better performance.

2.1 Traditional methods

First, let's look at the traditional models that have replaced the rule-based methods that were predominant at the time. These classifier methods had advantages over the rule-based techniques with respect to accuracy and stability (Q. Li et al., 2022).

2.1.1 Word embeddings

Before classical classifiers can be used for text classification, the text must be preprocessed to make it easier for machines to understand (Q. Li et al., 2022). There are several approaches to do this. TF-IDF, word2vec, and GLOVE are briefly described below and can be used here.

Term Frequency-Inverse Document Frequency (TF-IDF)

TF-IDF is a statistical method for evaluating the importance of a word in a document in relation to a collection of documents. The importance increases proportionately to the number of times a word occurs in the document but is balanced by the frequency of the word in the corpus. This helps to identify words that are important within a document, as opposed to those that are common (Baeza-Yates, Ribeiro-Neto et al., 1999).

word2vec

To obtain word vectors, word2vec uses local context information. This approach was developed by Mikolov et al. at Google (Mikolov et al., 2013). Neural networks train word vectors so that words with similar meanings are close to each other in the vector space.

The two models used in word2vec are Continuous Bag of Words (CBOW) and Skip-gram, as seen in Fig. 2.2. In the CBOW model, the input consists of several context words surrounding a target word in a sentence. These context words are represented as *w(t-2), w(t-1), w(t+1)* and *w(t+2)*. Each context word is converted into its vector representation using an embedding matrix, and these vectors are summed to form a combined context vector. This combined vector is then used to predict the target word *w(t)* by computing a probability distribution over the vocabulary. The model is trained to minimize the difference between the predicted and actual target words.

In the skip-gram model, the input is a single target word *w(t)*. This word is converted into its vector representation, which is then used to predict the vectors of the context words around it, represented as *w(t-2), w(t-1), w(t+1), and w(t+2)*. The model computes a probability distribution over the vocabulary for each context position to predict the most likely context words. The model is trained to minimize the difference between the predicted and actual context words.

The CBOW model predicts a target word from multiple context words by summing their vectors, while the Skip-gram model predicts multiple context words from a single target word. Both models learn word embeddings that capture the semantic relationships between words. Still, they approach the task differently: CBOW focuses on context-to-word prediction, while Skip-gram focuses on word-to-context prediction.

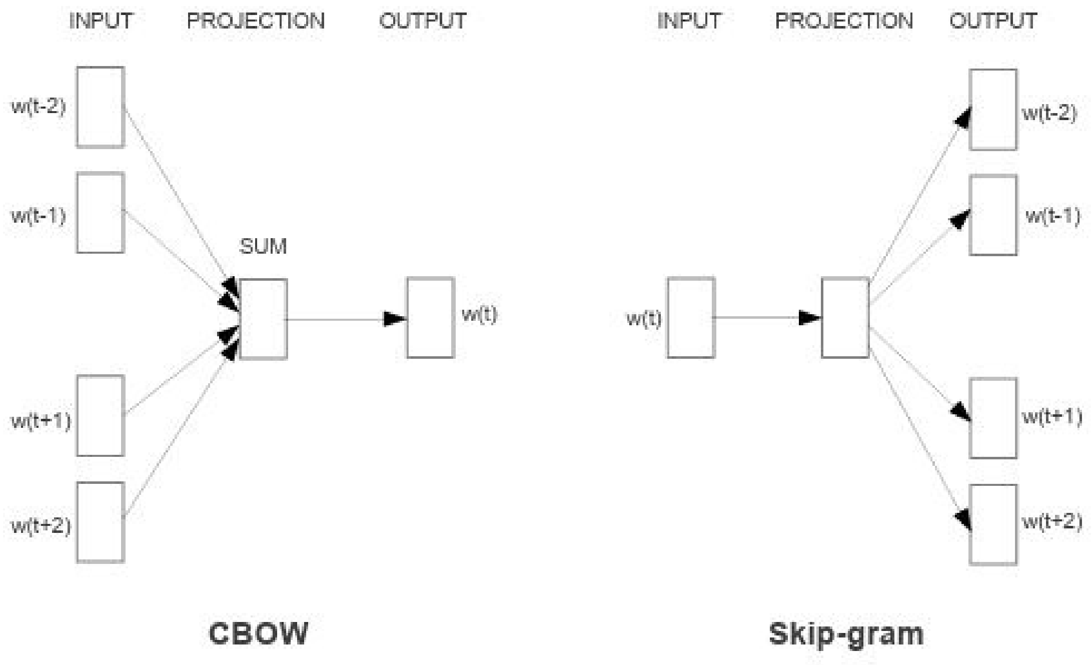


Figure 2.2: CBOW and Skip-gram (Mikolov et al., 2013)

GloVe

GloVe (Pennington et al., 2014) is a global log-bilinear regression model that combines global matrix factorization with local context window computation. It trains word vectors to make their product meaningful regarding the probability of two words occurring together, resulting in highly informative vector spaces.

2.2 Classifier

Once the texts have been prepared, classifiers can perform the actual text classification. All standard classifiers, such as Decision Trees, SVMs, KNNs, Random Forests, etc., can be used.

In the paper "The Text Classification of Theft Crime Based on TF-IDF and XGBoost Model" (Qi, 2020), the authors compare various well-known classifiers such as KNN, Naive Bayes, SVM and compare them with the very popular XGBoost classifier. The XGBoost classifier has become very popular in the recent past and has been found to provide the best results for many classification tasks. XGBoost (Chen & Guestrin, 2016) is a classifier specifically designed for efficiency and accuracy in gradient boosting. It features special techniques such as regularised learning targets and sparsity-aware algorithms to prevent overfitting and deal effectively with sparse data. In this implementation, the theft data should be categorized into one of the seven categories, such as burglary, car theft, etc. During preprocessing, the TF-IDF model was used to extract the features. The classifiers mentioned above were then used to assign the texts to a category. In this task, the authors also proved that the XGBoost classifier outperformed the other classifiers and achieved the best results.

A comparison is also made by Shah et al. (Shah et al., 2020) to compare different classifiers. This work develops a news text classification system for the BBC (British Broadcasting Corporation). TF-IDF is also used for text representation and preprocessing. Logistic Regression, Random Forest, and K-Nearest Neighbours are compared. The work is divided into text preprocessing (removal of stop words, etc.), text representation (TF-IDF), classifier implementation, and the classification itself. The evaluation shows that the Logistic Regression model performs best, ahead of Random Forest and KNN. The Logistic Regression model is the most stable, especially for smaller data sets.

Only two comparisons have been summarised here. The literature search revealed that almost all known classifiers have already been used in text classification.

2.3 Deep learning methods

Since the 2010s, the text classification problem has increasingly been tackled using deep learning methods instead of traditional models. A major advantage of these deep learning models is that they learn very well without domain knowledge (Q. Li et al., 2022). These models are, therefore, discussed in more detail below. CNNs, RNNs, LSTM, etc., and hybrid versions are presented, as well as how they master the task of text classification.

2.3.1 RNN & LSTM

Recurrent Neural Networks (RNNs) (Sherstinsky, 2020) are primarily used to capture long-range dependencies through recurring computations. They are networks that process information over time by storing and reusing states from previous computations. They are particularly good at tasks where the context or sequence of data is essential. This is especially important in speech recognition or predicting time series. But they can also be used for text classification. However, long-term dependencies are a major problem for RNNs, as this information is lost over the many steps.

Long Short-Term Memory Networks (LSTM) (Hochreiter & Schmidhuber, 1997), a subtype of RNNs, were developed to address this problem. This approach was designed to solve the problem of disappearing gradients, which complicates the learning of long-term dependencies in conventional RNNs. This is done with the help of "gates," which regulate which information should be added, removed, or retained. This makes it possible to effectively store both short- and long-term dependencies (Sherstinsky, 2020).

LSTM has already been used in the domain of IT ticket classification (Paramesh, 2023). Word embeddings are used to encode text descriptions of the tickets numerically. These are then used to train an LSTM model that can recognize long-term dependencies in the data. A data set of around 10.000 tickets is used for training. The tickets must be assigned to one of eighteen categories. The LSTM model is compared with classic classifiers (SVM, Logistic Regression, Naive Bayes, and KNN). It turns out that the LSTM model performs significantly better than these and achieves an accuracy of 90.8%. The same author also trains a CNN on the same data set. This work is presented in the next section; CNN achieves an accuracy of 91.2%.

2.3.2 CNN

Convolutional Neural Networks (CNN) (Albawi et al., 2018), initially developed for image processing, process the input in multiple layers. These start with convolutional layers that extract specific features using learnable filters, followed by activation functions to introduce non-linearity. There are also pooling layers, which reduce dimensionality while retaining important information. Finally, there are fully connected layers, which process the extracted features to produce the final output. This can be, for example, classification labels. However, as later work has shown, CNN can also be used for text classification.

In his work, Kim laid the foundation for TextCNN (Kim, 2014), which later became the basic model for many implementations of CNN for text classification. It is shown that excellent results can be achieved in text classification by fine-tuning hyperparameters and static vectors. In addition, a change in the architecture is proposed to allow task-specific and static vectors.

A concrete implementation (Paramesh & Shreedhara, 2022) of CNN for classifying IT service desk tickets shows how this approach can be used for automatic ticket classification. Word embeddings are used to represent the ticket description and automatically extract the features that are important for classification. The output classification layer then uses the extracted features to predict the ticket category. This model was then evaluated against traditional models such as SVM, Naive Bayes, Logistic Regression, or KNN. It turned out that CNN outperformed these models. In total, there were eighteen categories to which a ticket could belong. CNN was able to predict the correct category with 91.2% accuracy. SVM, the best classic classifier, had an accuracy of around 87%.

Fig. 2.3 illustrates the different approaches of RNNs and CNNs to processing text for NLP tasks. On the left, the RNN model processes text sequentially, with each word passing through the network to produce a hidden state that captures the dependencies between the current word and previous words. These hidden states *(h_1, h_2, h_3, h_4, h_5)* are connected in a chain, with the last hidden state being used to predict the label, demonstrating how RNNs maintain the context of the whole sequence.

On the right, the CNN model processes the text using convolutional layers that apply filters to the word vectors to extract local features. These features are then pooled to reduce dimensionality, and the pooled features are concatenated to form a comprehensive representation of the text. This representation predicts the label, demonstrating how CNNs effectively capture local patterns. Each architecture has its strengths, with RNNs better at handling sequential dependencies, as shown by the connected hidden states, and CNNs more efficient and effective at capturing local patterns through convolution and pooling layers (Q. Li et al., 2022).

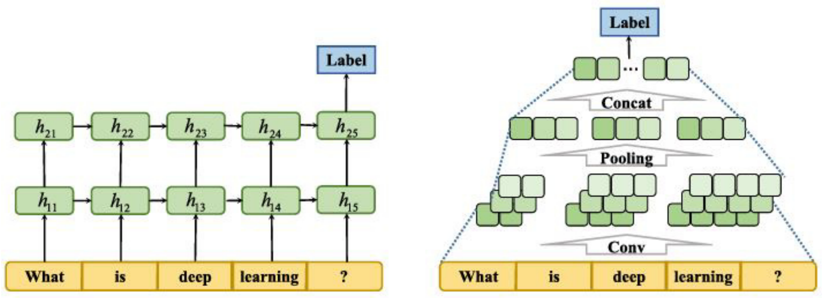


Figure 2.3: Comparison RNNs (left) and CNNs (right) (Q. Li et al., 2022)

2.3.3 Pre-trained

Before moving on to the transformer architecture, which is the main focus of this paper, we will look at a model that, like GPT and BERT, is pre-trained. Embeddings from Language Models (ELMo) was developed by Peters et al. (Peters et al., 2018). ELMo generates word embeddings that capture the characteristics of word usage and its variations in different linguistic contexts. The word embeddings already discussed, such as word2vec or GloVe, use a fixed vector representation for each word. ELMo, on the other hand, is based on the internal states of a deep bidirectional language model (biLM) pre-trained on a text corpus. Precisely, it consists of LSTM networks running over the text in both forward and backward directions. This makes it possible to create a context-dependent embedding for each word occurrence in a text.

3

Transformer

Having looked at some well-established approaches to text classification, we look at a relatively new approach in this chapter. Firstly, we will introduce the Vaswani et al. transformer model (Vaswani et al., 2017). Then, we take a look at the two best-known implementations, the BERT model (Devlin et al., 2018) and GPT (Radford et al., 2018).

With the introduction of BERT and GPT, two game-changing models have been established in artificial intelligence. Despite their different approaches, both models have quickly gained wide acceptance in the research and AI community. Both models have proven that, although they are based on different approaches, they can be used effectively for text classification tasks.

3.1 Attention Is All You Need - Transformer

The paper "Attention Is All You Need" by Vaswani et al.(Vaswani et al., 2017) presents a new model that represents a departure from traditional methods in the field of NLP. As previously described, the prevailing models were based on complex recurrent or convolutional neural networks. In their work, they present a Transformer that is based solely on attention mechanisms and foregoes recursion and convolution. Because the model relies on its own attention, it can weigh the importance of different parts of the input data differently, resulting in more differentiated and context-aware representations. By utilizing self-attention, the Transformer can process all parts of the input data in parallel, drastically reducing training times and improving the model's ability to capture complex relationships within the data.

This process is described in more detail in Fig. 3.1. The left part shows the encoder and the right part shows the decoder. The encoder is responsible for processing the input sequence and generating the encrypted representation. First, the embedding layer converts the input text into continuous vectors. Since the Transformer model does not inherently process sequences in the correct order, positional encodings are added to the input embeddings to give the model information about the position of each word in the sequence. This helps the model to capture the order of the words. Then comes the actual encoder, which consists of

N identical layers, which in turn consist of two parts. The first is the multi-head attention. This layer lets the model focus on different parts of the input sequence simultaneously. Multiple attention heads work in parallel, each learning to focus on different aspects of the sequence. The output of the multi-head attention layer is added to the input of this layer and then normalized. The second part is the feed-forward network. Here, a fully connected feed-forward network is applied to the output of the attention layer. The decoder has a similar basic structure. However, it contains a masked multi-head attention. This layer is identical to the encoder's multi-head attention. Still, it includes masking to ensure that the prediction for a given position only depends on the known outputs at preceding positions. The final output from the decoder is passed through a linear layer followed by a softmax layer to generate a probability distribution over the vocabulary for the next word in the sequence.

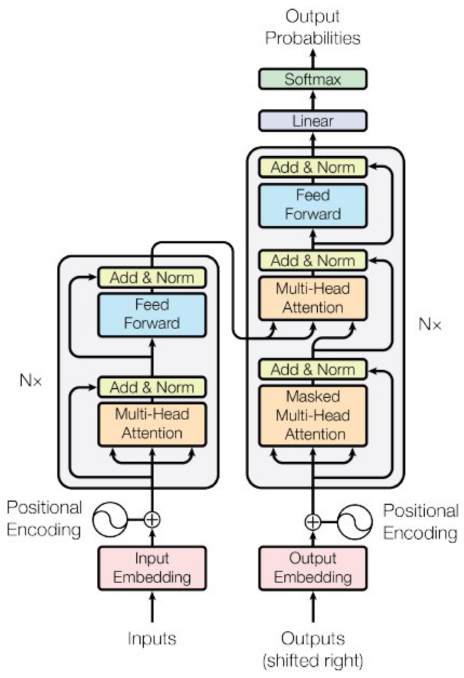


Figure 3.1: Transformer model (Vaswani et al., 2017)

The Transformer model outperformed the best previously published models in the WMT (Workshop on Machine Translation) 2014 English-German translation task by more than 2.0 BLEU (Bilingual Evaluation Understudy) points, setting a new record with a score of 28.4 BLEU. WMT is a competition and evaluation series that focuses on benchmarking and advancing the state-of-the-art in machine translation. BLEU is a widely used metric for evaluating the quality of machine-translated text by comparing it to one or more reference translations. It is designed to approximate human judgment and quantitatively measure how similar machine-generated translation is to professional human translations. In the WMT 2014 English to French translation task, the large version of the Transformer achieved a BLEU score of 41.0, outperforming all previously published individual models while incurring less than a quarter of the training cost of the previous best model. These results show that the Transformer model sets a new standard in machine translation by achieving significantly better results at a lower training cost (Vaswani et al., 2017).

3.2 BERT - Bidirectional Encoder Representations from Transformers

Devlin et al. developed the Bidirectional Encoder Representations from Transformers, or BERT for short, based on the transformer model described above. It was developed in cooperation with Google Research. It is still one of the most advanced natural language representation models and is very popular, especially in the scientific community. The BooksCorpus (800 million words) and the English Wikipedia (2,500 million words) were used for pre-training. This work gave rise to the BERT Base model (110 million parameters) and the BERT Large model (340 million parameters), which outperformed the prevailing models at the time on various tasks. This is mainly because, unlike models such as OpenAI GPT (Radford et al., 2018), which are based on a left-to-right architecture, the BERT representation can capture the left and right context. The architecture of BERT is briefly described below.

3.2.1 Embeddings

To make BERT usable for various downstream tasks, the input representation is designed to uniquely represent either a single sentence or a pair of sentences in a sequence of tokens. The term sentence refers to any sequence of related text, not necessarily a linguistically correct sentence. Sequence refers to the sequence of tokens that BERT receives as input, consisting of either a single sentence or two linked sentences.

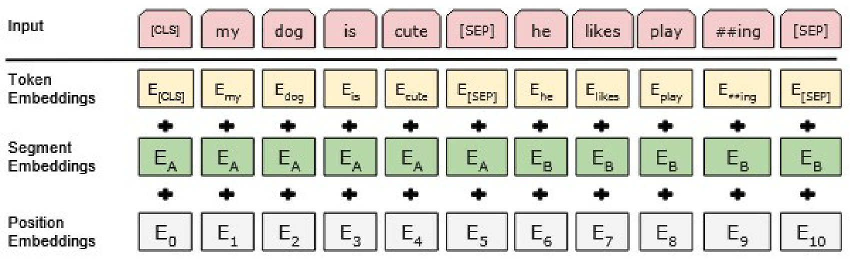

Input
[CLS]
my
dog
is
cute
[SEP]
he
likes
play
##ing
[SEP]
Token Embeddings
$E_{[CLS]}$ E_{my} E_{dog} E_{is} E_{cute} $E_{[SEP]}$ E_{he} E_{likes} E_{play} $E_{\#\#ing}$ $E_{[SEP]}$
Segment Embeddings
E_A E_A E_A E_A E_A E_A E_B E_B E_B E_B E_B
Position Embeddings
E_0 E_1 E_2 E_3 E_4 E_5 E_6 E_7 E_8 E_9 E_{10}

Figure 3.2: BERT - Embeddings (Devlin et al., 2018)

The graphic (Fig. 3.2) shows how BERT processes input data. For each input, BERT creates a unique representation consisting of three parts:

- **Token embeddings:** These are representations of each word or punctuation mark in the input. Each word is represented by a vector derived from its meaning in the context of the whole vocabulary.

- **Segment embeddings:** These show which sentence (A or B) each word belongs to. This is useful when comparing or relating two sentences, as in a question-and-answer task.
- **Position embeddings:** Since BERT does not take word order into account like traditional language models, it needs a way of knowing the position of each word within the sentence. Positional embeddings provide this information.

The overall representation of a word in BERT is the sum of these three embeddings. The unique [CLS] (short for "classification") token is used to aggregate information from the entire input sequence. It is added at the very beginning of the input sequence. The [SEP] token (short for "separator") is used to separate different segments or sentences within the input sequence. It helps the model distinguish between multiple segments in tasks that involve pairwise inputs, such as question-answering or sentence pair classification.

3.2.2 Pre-training

BERT was pre-trained using two unsupervised tasks:

Masked Language Model

The first task is the so-called Masked Language Model (MLM). As mentioned above, BERT is a bi-directional model. Unfortunately, conventional language models are limited to training in a single direction, either left to right or right to left. Bidirectional conditioning would allow each word to see itself.

To develop a more complex bidirectional understanding, the MLM technique is used, where a certain proportion of the input tokens are randomly obscured, and the model is asked to predict them. 15% of all word price tokens in each sequence are randomly masked with the usage of a MASK token. Predictions are made based on the context of the unmasked tokens, using a probability distribution over the possible tokens. The focus is on predicting the masked tokens.

However, this results in a mismatch between pre-training and fine-tuning, as the MASK token does not occur during fine-tuning. Therefore, not all masked words are replaced by the MASK token, but 15% of the words are randomly selected. 80% of these selected words are replaced by a MASK token, 10% by a random word, and the remaining 10% are left unchanged. The goal is to predict the original word based on these changes correctly.

Next Sentence Prediction

The second task is the Next Sentence Prediction (NSP). NSP is designed to help models understand the relationship between two sentences and is not learned directly by traditional language modeling. With NSP, models are trained with pairs of sentences labeled "IsNext" if the second sentence logically follows the first or "NotNext" if there is no connection. This training setup uses a simple approach where, for each training example, there is a 50% chance that the second sentence is the actual successor of the first sentence in the corpus and a 50% chance that it is a random sentence from elsewhere in the corpus. Despite the simplicity of this method, it significantly improves the model's performance in question answering and natural language inference by enhancing the understanding of sentence relationships (Devlin et al., 2018).

3.3 GPT - Generative Pre-Trained Transformer

In their paper "Improving Language Understanding by Generative Pre-Training," Radford et al. (Radford et al., 2018) present their model, Generative Pre-Trained Transformer (GPT). In their work, they show that great progress can be made in natural language understanding tasks through generative pre-training on a corpus of unlabeled text, followed by fine-tuning to the specific task. GPT was developed by the company OpenAI. Open AI is an AI research and deployment company ('OpenAI', 2024). GPT was trained using the BooksCorpus dataset, which contains 7.000 unpublished books. GPT-1 contains 117 million parameters and, at the time of publication, achieved top performance in a wide variety of tasks. The training consists of two stages. In the first stage, a capable language model is trained on a large text corpus. In the second stage, fine-tuning, the model is trained on a specific task with labeled data.

Unsupervised pre-training

This step aims to maximize the likelihood of a token given its previous context by using a multi-layer transformer decoder that employs a self-attention mechanism and position-wise forward layers. This model predicts the probability distribution of the next token. The Transformer learns this through unsupervised learning by optimizing its parameters through stochastic gradient descent. This process prepares the model to understand and generate language before fine-tuning it for specific tasks.

Supervised fine-tuning

After pre-training the language model on a large unsupervised text corpus, the model is fine-tuned in a supervised task with a labeled data set. The pre-trained model processes the input tokens to obtain a representation that is then used by an additional linear layer to predict the label. Language modeling is kept as an auxiliary target to improve performance and convergence during fine-tuning. In the fine-tuning phase, only some additional parameters are added, such as the weights of the initial linear layer and the embeddings for special delimiters. This process effectively uses the general language understanding learned during pre-training for specific tasks.

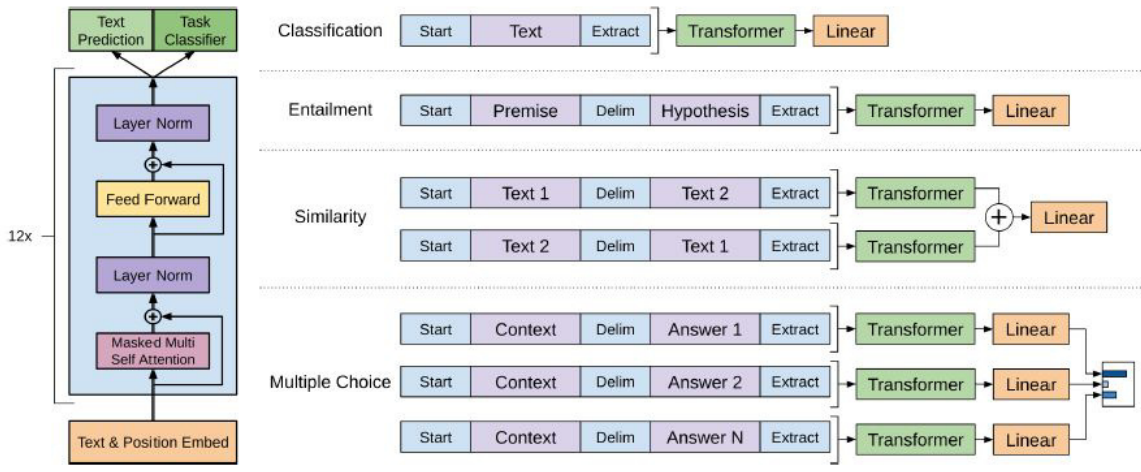


Figure 3.3: GPT - Architecture and fine-tuning process (Radford et al., 2018)

Figure Fig. 3.3 describes the method for fine-tuning a GPT model for various natural language processing tasks, such as text classification, textual entailment, and question answering. The left part of the image illustrates the architecture of the Transformer model, which is composed of layers that include normalization, feed-forward networks, and self-attention mechanisms. The right part of the image shows how various task-specific structured inputs are transformed into a token sequence format that the pre-trained GPT model can process. This approach allows the same pre-trained model to be used for different tasks without extensive changes to the architecture by simply adding start and end tokens and converting the structured inputs into sequences that the model understands.

Based on GPT-1, the model has been continuously developed. In 2019, GPT-2 was released, which already contained 1.5 billion parameters. GPT-3 was a further leap with 175 billion parameters and was introduced in 2020. Its successor, GPT-3.5 Turbo, was introduced in 2022. This was the first time GPT became known to the general public, as it allowed access to ChatGPT. The latest version at this time is GPT-4, which was released in 2023 ('OpenAI', 2024).

Fig. 3.4 illustrates the structural and functional differences between BERT, GPT, and ELMo. BERT employs bidirectional Transformers to capture comprehensive context, GPT uses unidirectional Transformers for generative tasks, and ELMo leverages bidirectional LSTMs to encode rich contextual information. BERT takes into account the context to the left and right of the target word when creating word embeddings. GPT, on the other hand, uses a left-to-right transformer. ELMo trains LSTMs independently of left-to-right and right-to-left to generate the embeddings (Devlin et al., 2018).

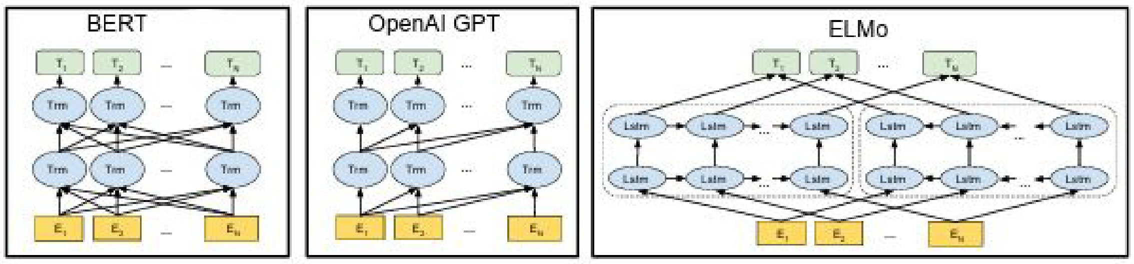


Figure 3.4: Comparison pre-trained models, slightly adapted from (Devlin et al., 2018)

As this literature review has shown, LLM is currently considered the state-of-the-art method for text classification. They not only achieve better performance but also impress with their simplicity. Based on these findings, it was decided to abandon traditional classification and DNN and to use LLM for ticket classification at Doka GmbH.

4

Fine-tuning of GPT and BERT for Text Classification

In recent years, fine-tuning has been an effective approach to training models for specific tasks, even with little labeled data. The models are first pre-trained on as broad a basis as possible and then fine-tuned to the labeled data using gradient descent methods. Furthermore, it has already been shown that fine-tuning can outperform the strategy of fixing the representation in the source task (Shachaf et al., 2021). This chapter shows how BERT and GPT deal with the task of text classification. It explains whether there is already scientific work on fine-tuning the models. It shows what gaps still exist in current research and lays the foundation for how the models will be built later.

4.1 BERT

In their paper "How to Fine-Tune BERT for Text Classification?" Sun et al. (Sun et al., 2019) gives the most comprehensive guide on how to train a BERT model from scratch. They examine in detail different fine-tuning methods to optimize the performance of BERT in the task of text classification. They propose a three-step approach to train BERT further:

- **First step:** Further pre-training of BERT to generate task-specific or domain-specific knowledge. This should lead to the model's better understanding of the nuances in a particular domain.
- **Second step:** Optional multi-task fine-tuning, where the model is trained on different tasks. This shared knowledge should lead to an improvement in performance.
- **Third step:** Actual fine-tuning, where the model is given concrete examples with the respective known labels so that it can be trained specifically for the task.

"A Fine-Tuned BERT-Based Transfer Learning Approach for Text Classification" (Qasim et al., 2022) shows how the fine-tuning of BERT can be used to classify text. In this specific case, the goal is to identify fake news in tweets related to COVID-19. Different models like BERT-base, RoBERTa, DistilBERT, or ALBERT are used as a base model. These are pre-trained models. In the next step, they are fine-tuned to the specific data set, in this case, tweets. The results show how vital the fine-tuning step is, as it can significantly improve the results. All models can achieve excellent results in this way. The accuracy of detecting fake news is around 99% for all models.

RoBERTa (Y. Liu et al., 2019), DistilBERT (Sanh et al., 2019), and ALBERT (Lan et al., 2019), which were mentioned in the paragraph above, are variants of the original BERT model. All three variants are further developments of the original model to improve aspects such as efficiency, speed, or performance. RoBERTa uses a more extensive data set as part of the training process and omits the NSP task performed in the original BERT. This leads to an increase in model performance on various NLP tasks without fundamentally changing the architecture of BERT. DistilBERT is a smaller and faster version of BERT. It is 40% smaller but still has 97% of the language understanding capability of BERT. This is achieved by "distillation", where knowledge from the large BERT model is transferred to a smaller model. This means that training can be carried out more quickly. ALBERT takes a different approach to reducing the size of the model by reducing the number of parameters. This also results in a lighter model that can be trained more quickly.

In their work, Liu et al. create Ticket-BERT (Z. Liu et al., 2023), a BERT model fine-tuned to classify Incident Management tickets. They use 76.000 tickets from Microsoft Kusto that belong to one of the ten defined categories. The dataset is divided into three different types, namely human-written descriptions, machine-generated descriptions, or a mixture of both. A separate model is generated for each. As part of the fine-tuning process, BERT is fine-tuned to be trained specifically for this classification. The trained model is then validated using classifiers such as Naive Bayes or Logistic Regression. The evaluation shows that the fine-tuned model outperforms the other models. A precision of over 98% is achieved for the machine-generated tickets. For the human-generated tickets, it is around 86%. This work also shows that the unambiguous identification of subject and content leads to a significant increase in performance.

As has been shown, BERT is particularly popular in the scientific world for text classification. Numerous examples show how BERT has been fine-tuned to specific data sets. This is mainly because BERT is open source and can, therefore, be used by anyone free of charge. It was one of the first models to be made available.

Although Sun et al. show in their work a very detailed plan of how BERT can be optimally trained on a specific text classification task, this work does not follow this plan 1:1. The reasons for this are manifold. The focus of this paper is to compare GPT and BERT. Since the pre-training and multi-task fine-tuning of BERT, as proposed in the work of Sun et al., is not possible with GPT in this form, this would lead to an unequal starting point. It would also require a data set allowing us to do the pre-training. It could also be argued that a variant of BERT, such as RoBERTa, could make the model faster or more powerful.

However, as mentioned, this work focuses on comparing GPT and BERT, aiming to create the same primary conditions as far as possible. In addition, the focus is always on making the system so that it can be reused by Doka GmbH later and easily retrained.

4.2 GPT

An example of how GPT, specifically GPT-3 in this paper, can be used for text classification is provided by Pawar et al. (Pawar & Makwana, 2022). They use the Marathi Polarity Labeled Corpora, a tourism dataset. This consists of 75 positive and 75 negative reviews. A BERT model is fine-tuned for comparison. They use 80% of the dataset as training data. The remaining 20% is used for testing. In contrast, the GPT-3 model is not fine-tuned. GPT models are known to be few-shot learners. Therefore, they only use 20% of the data set and use the remaining data for testing. The GPT result is impressive, achieving 98% accuracy on this dataset. In comparison, the fine-tuned BERT model achieves a precision of 80%. This paper shows that GPT performs exceptionally well when only a small amount of data is available for training.

Another paper (Mayer et al., 2023) also uses GPT for text classification. Here, prompt engineering is discussed in more detail, i.e., how to transfer the labeled data to the model. A GPT-J-6B model (a downloaded version of GPT) was used with a size of 825GB. The classification aims to show whether an email is polite or not. To do this, people labeled a dataset of 2.088 emails. 70% of them were used for training, the rest for testing. The prompt-based approach of GPT was compared with a fine-tuning approach of BERT. The fine-tuned BERT model achieved an accuracy of 92%. In contrast, the GPT model achieved an accuracy of 93%.

Zhao et al. (B. Zhao et al., 2023) show in their work that ChatGPT can be used for text classification. They show this in the context of agricultural text in different languages. They argue that fine-tuning is highly dependent on the training data and that such pre-trained language models have poor domain transferability. Furthermore, the complexity of such large fine-tuned models should not be underestimated. They use GPT-3.5 and already GPT-4 to show that zero-shot learning is also possible. This means that no training data and labeled training data are provided. The results show that ChatGPT delivers comparable or sometimes even better results than fine-tuned models.

Despite an intensive literature search, finding scientific papers where GPT is used for text classification using fine-tuning was impossible. However, in a work (Lee & Hsiang, 2020), for example, a GPT model is fine-tuned so that it can automatically generate patent claims. Patent claims have a particular language, so developing them with conventional LLMs is challenging. For the fine-tuning, over half a million patent claims were therefore used for training. With the help of fine-tuning, it was possible to create patent claims very similar to those produced conventionally.

As shown in this section, there is little to no literature where GPT has been fine-tuned to optimize it for text classification. There are undoubtedly many reasons for this. It turns out that GPT is a zero-shot or few-shot learner. This means little or no labeled training data is needed to classify text. However, it should be noted that in the literature, classification is usually done on only two classes, which are usually very general, such as "polite" or "not polite". However, in the task of classifying Doka GmbH tickets, it is sometimes necessary to distinguish between over a hundred classes. Moreover, these classes are not generic but have been defined by Doka GmbH. In addition, some attempts have been made to classify the Doka GmbH tickets using a few-shot approach. However, the results showed that the tickets were too diverse to be labeled correctly by a conventional LLM. Another point is that the GPT models that can be fine-tuned are still relatively new. The models have only recently become known to the general public, and access to them was very limited before GPT-3. Of course, the cost should not be underestimated. Now that BERT is free to use, there is a large number of scientific studies. There will probably never be such a rush for GPT from the scientific community simply because of the cost barrier. The reason for this paper is to help show whether it makes sense to fine-tune GPT at all and whether the performance compared to BERT makes it worth paying for.

5

Data understanding & exploration

Before we can start training the models, getting a proper overview of the data sets is essential. This section, therefore, involves intensive data understanding and exploration. It starts with a general part and is then divided into the respective five data sets.

5.1 Overview

Doka Gmbh provided five data records, divided into the five target variables: Priority, Form, Queuename, Servicetype, and Serviceline. The slightly different size is because the respective label was unavailable for every ticket. To be allowed to use the data records, Doka Gmbh IT cleaned up and anonymized them in advance.

The tickets from the years 2022-2023 were used. As these also have to be sent to Open AI as part of the fine-tuning of GPT, removing or anonymizing sensitive data was critical. For this reason, confidential information such as telephone numbers and IP addresses were replaced with random numbers during pre-processing. Email addresses and names were also replaced with user@website.com and names with John Doe. Unnecessary information, such as HTML tags, email signatures, etc., was removed to create a clean dataset.

In Table 5.1, one can see the format in which the tickets have been made available and how they have been used for training. It consists of the "Prompt" and the "Completion". The prompt contains the heading and the ticket content, separated by an \n. The "Completion" column contains the class to which the ticket belongs.

Table 5.1: Ticket example

	Prompt	Completion
Original	Subject: Email Issue: Not Receiving Emails\nContent: Hello, I currently face some issues with my Outlook. A colleague just sent me an email, but they don't appear in my mailbox. Could you please check if everything is fine with my email address, philipp.engelbrechtsmueller@doka.com? Best regards, Philipp Engelbrechtsmüller\n	10
Cleaned	Subject: Email Issue: Not Receiving Emails\nContent: Hello, I currently face some issues with my Outlook. A colleague just sent me an email, but they don't appear in my mailbox. Could you please check if everything is fine with my email address, user@website.com? Best regards, John Doe\n	10

The classes were mapped to numerical values (Listing 5.1). The ticket text itself includes a defined "subject", which is terminated with a \n.

Listing 5.1: Mapping target variables

```
{
" 0": "Emergency",
" 1": "High",
" 2": "Normal",
" 3": "Longtime"
}
```

In the end, five data sets, each containing around 107.000 tickets, are available, as can be seen in Table 5.2.

Table 5.2: Overview of the target variables

Target variable	Total Dataset	Classes	Values
Priority	107.772	4	Emergency, High, Normal, Longtime
Form	107.236	100	Newuser_SAP, No Form, etc.
Queuename	107.708	6	Changemanagement, Incident, etc.
Servicetype	107.304	116	SAP, PointOut, Prolag, etc.
Serviceline	107.410	72	IT-DevOps, PCM 1st Level Support, etc.

Queuename was used to explore the entire data set. This information also applies to all other data sets as they differ only slightly in number.

The first step was to analyze the distribution of languages. Fig. 5.1 shows how each language gets distributed across the whole dataset. A total of 57.944 tickets in this dataset are in English. 46.817 tickets are in German. The remaining 2.947 are in languages such as Italian and Spanish, or the langdetect library used could not determine the language. As the selected models, which will be fine-tuned later, are multilingual, all languages are used for training.

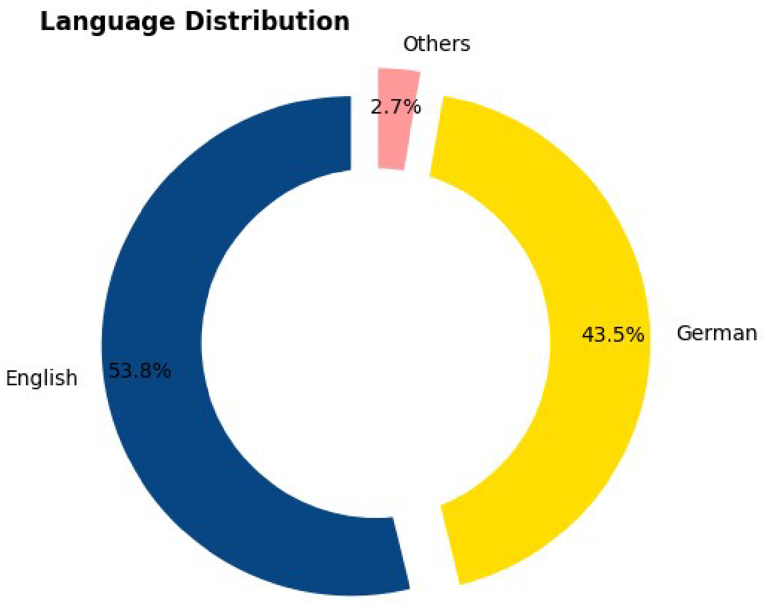


Figure 5.1: Language distribution

The next step was to analyze how many words (Fig. 5.2) each ticket contains. It turns out that the majority of tickets have less than 50 words. About 30.000 tickets have between 50 and 100 words. The frequency of tickets decreases steadily as the number of words increases. It is essential that employees keep their tickets short and to the point. This saves the IT helpdesk staff a lot of time, and, as we will see later, the models will also find it easier if the tickets are not filled with too much irrelevant information.

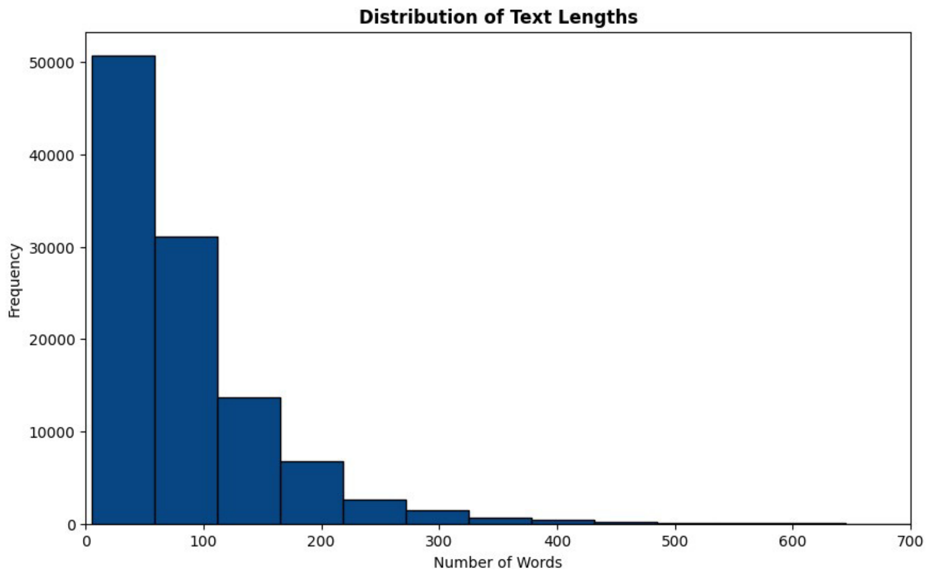


Figure 5.2: Text length per ticket

The individual datasets are discussed in more detail below. A general overview of what the target variable means is given. We will also look at the distribution of the classes.

5.2 Priority

The first target variable to be predicted later is the "Priority" value. It specifies the priority with which the tickets are to be processed. There are four classes: Normal, Longtime, Emergency, and High. Longtime tickets usually involve a change request, meaning a process or program is to be adopted. This is generally associated with a longer processing time. Tickets with the priority Emergency or High are usually incidents, which means an acute error needs to be rectified immediately. Normal tickets are standard processes that involve creating users, adjusting authorizations, etc. This data set contains a total of 107.772 tickets.

Table 5.3: Class distribution Priority

Class	Count
Normal	95.352
Longtime	10.479
Emergency	1.426
High	515

The data set is quite unbalanced, as seen in Table 5.3 and Fig. 5.3. Almost 90% of the tickets fall into the Normal class. Longtime tickets are in second place. Very few tickets are considered incidents and fall into the Emergency or High category.

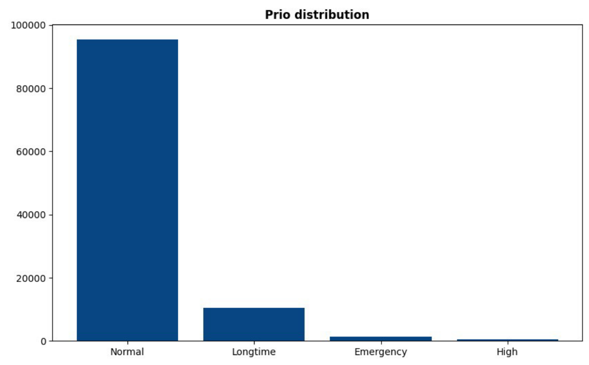


Figure 5.3: Distribution Priority

5.3 Form

The second data set is dedicated to the variable "Form". A Form can be understood as a checklist. Forms must be used for specific processes such as creating SAP users, general

DOKA users, or posting messages on the intranet. This ensures the process has been carried out correctly and can be checked retrospectively using the checklists. The entire data set consists of 107.236 tickets. There are 100 different classes for this attribute. One class is the "no form" class, which can be understood as a NULL value. As can be seen in Fig. 5.4, exactly 20% of the tickets have a form, and the rest are without.

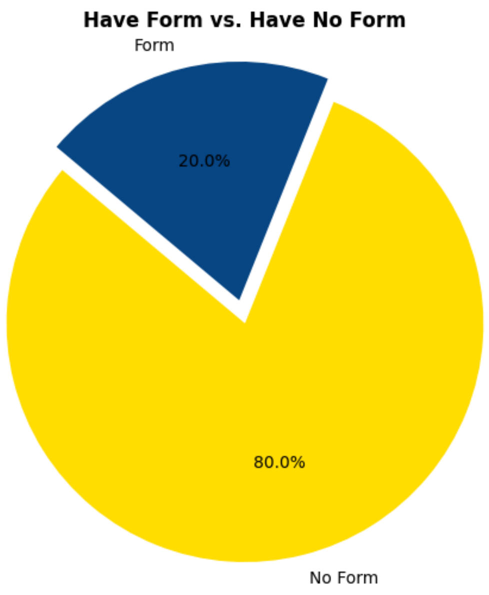


Figure 5.4: Form vs. No Form

In Fig. 5.5, you can see a list of the TOP10 categories within the tickets that have a Form at all — the majority deal with the activation or deactivation of users.

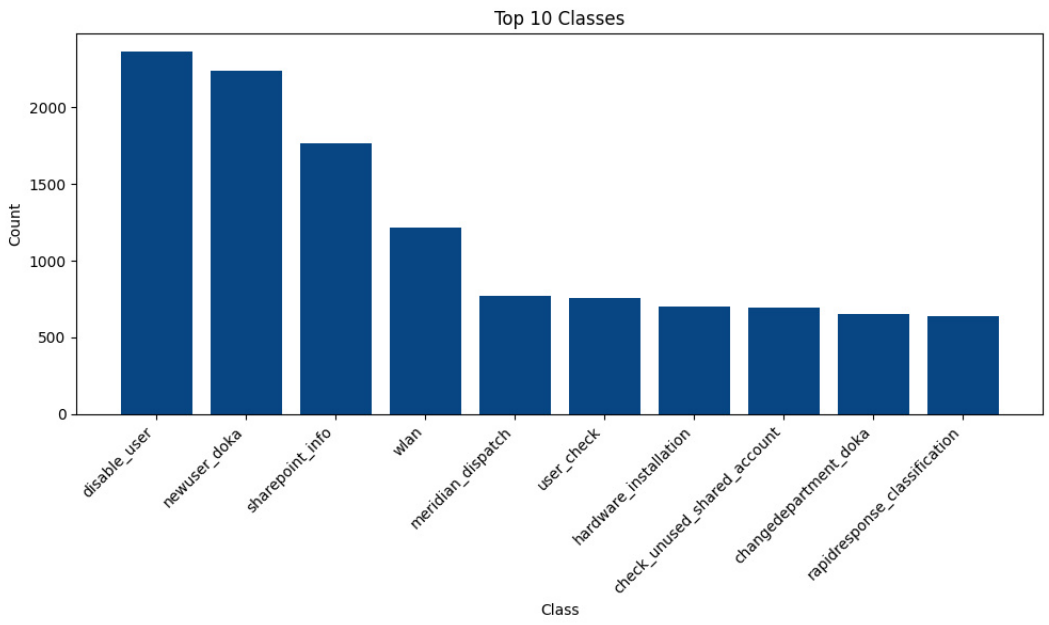


Figure 5.5: TOP10 Form classes (if they have a form)

5.4 Queuename

The third variable is "Queuename," and the data set consists of a total of 107.708 tickets, which are divided into six categories. The categories aim in the same direction as those of "Priority". They are mainly used to differentiate between change management and incidents. In addition, some tickets are used for orders and are therefore labeled with the Queuename "Order". In the meantime, Doka GmbH has adapted the categories to provide more meaningful results. However, these six classes are still used in this thesis.

Table 5.4: Class distribution Queuename

Class	Count
Changemgmt	80.771
Incident	18.639
Order	6.402
Ittask	1.538
Archive	317
Voice	41

As can be seen in Table 5.4 and Fig. 5.6, the vast majority of tickets are assigned to the change management category. In second place are tickets that are labeled as Incident. The rest are divided between Order, Ittask, Archive, and Voice.

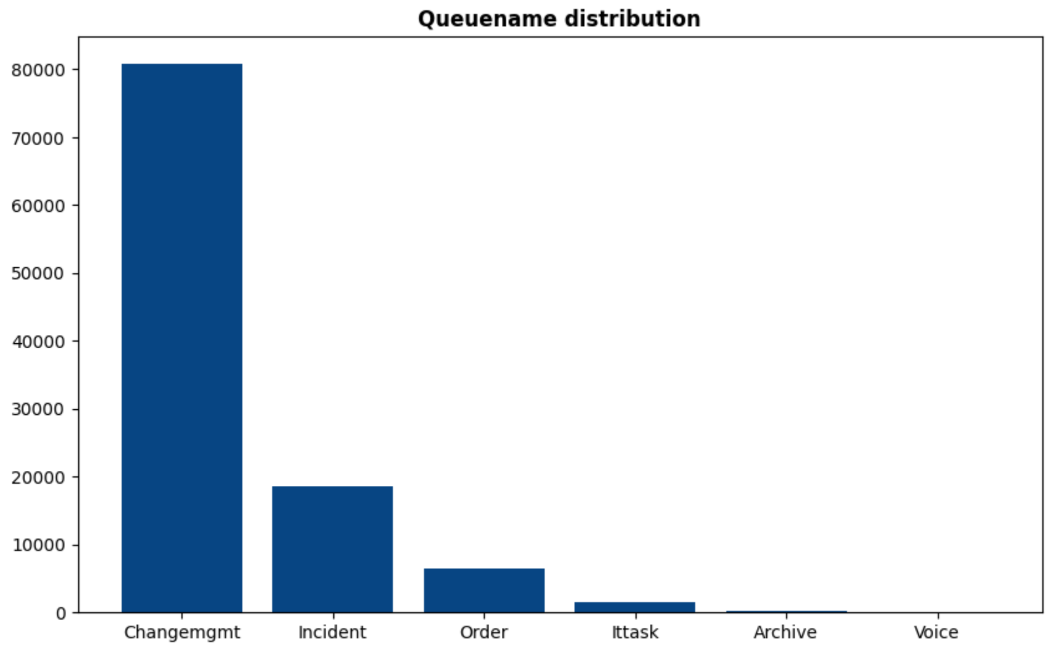


Figure 5.6: Queuename distribution

5.5 Servicetype

Now let's look at one of the two most essential target variables: "Servicetype". The data set consists of 107.304 tickets and a total of 116 classes. The data set describes the topic or program to which the ticket is dedicated. This can be a software program or a general topic such as hardware, authorizations, etc. This target variable is so important because it provides essential information for reporting. It shows in which subject areas most questions occur, and this information can be used to make proactive improvements.

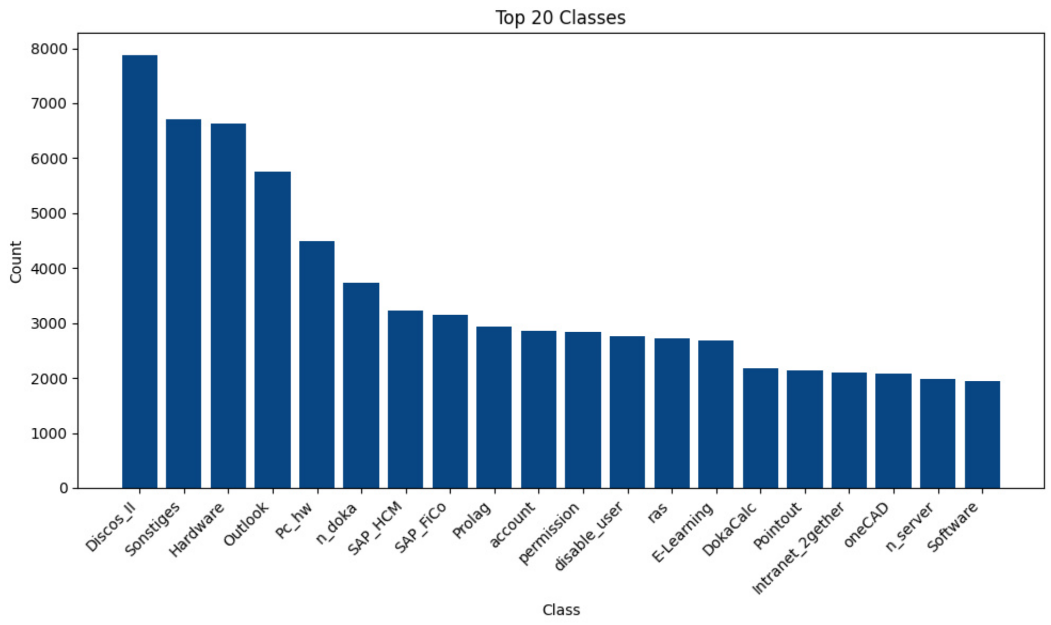


Figure 5.7: TOP20 Classes Servicetype

The data set is quite balanced, as Fig. 5.7 shows. Most cases are found in DISCOS_II, an internal Doka software program. Other classes that occur frequently are Sonstiges, Hardware, and Outlook.

5.6 Serviceline

The most important attribute of a ticket is the "Serviceline." It indicates which department or team is responsible for the ticket. They must process the ticket, offer the internal customer a solution, and close it. The data set consists of 107.410 tickets divided into 72 Servicelines. As can be seen in Fig. 5.8, almost half of the tickets are processed directly by the helpdesk. The IT helpdesk is effectively responsible for first-level support. If specialized knowledge is required to process the ticket, the ASP (Ansprechpartner) is usually responsible. This is the case for the other half of the tickets.

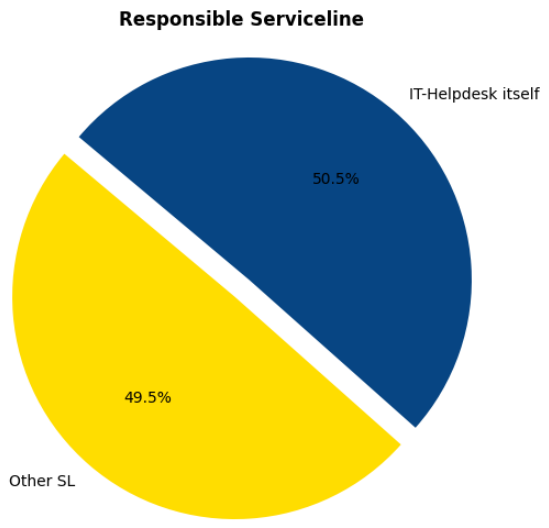


Figure 5.8: Distribution Serviceline

In Fig. 5.9, the IT-HD Serviceline is excluded to provide a better overview of the remaining Servicelines. The IT teams are represented at the front of this list. They are followed by the teams responsible for special programs such as DISCOS, the respective SAP modules, and other programs.

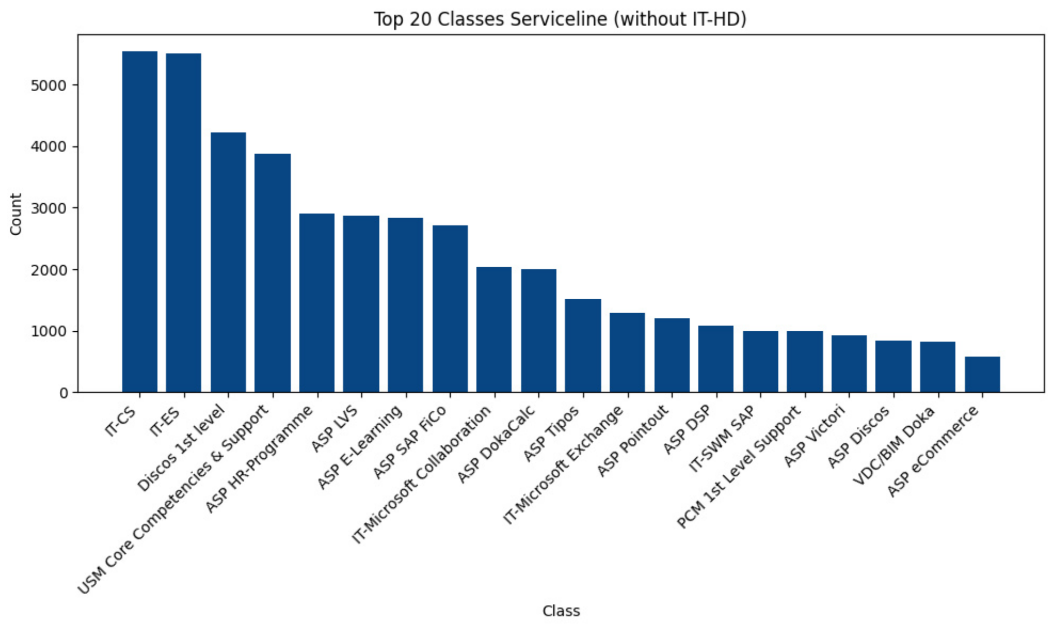


Figure 5.9: TOP20 Classes Serviceline (without IT-HD)

As this in-depth look at the data has shown, there are some data sets that are very imbalanced. Nevertheless, these data sets are used to train the models. This shows whether BERT or GPT can handle a small number of training examples better. GPT is advertised as a few-shot or zero-shot learner. Training with this unbalanced data set should confirm or negate this statement. How to improve the imbalance of the data sets is discussed in Chapter 8.

6

Implementation of fine-tuned GPT and BERT model

Now that we have examined the data sets more closely, it is time to fine-tune the LLMs.

At the request of Doka GmbH, a separate model was trained for each target variable. The following number of tickets was used to train the models:

Table 6.1: Training tickets for each model

Model	Tickets
Form	101.874
Priority	102.383
Queuename	102.322
Serviceline	102.039
Servicetype	101.938

The work first deals with the BERT model and then with the GPT model.

6.1 Fine-tuning of BERT

As already mentioned, a separate model was fine-tuned for each attribute. The training settings are briefly explained in the following, before the exact implementation is discussed in detail. The implementation is based on the recommendations of Devlin et al. and Sun et al.

6.1.1 Training setting

As IT governance had not given the go-ahead to train the BERT model using a cloud provider, it had to be trained locally on the company laptop. Unfortunately, it was also not possible to use Nvidia CUDA[1]. This would have allowed the use of the GPU, which is

[1]https://developer.nvidia.com/cuda-toolkit

usually used for this type of work. So, the training was done on the CPU, an Intel Core i7-10750H CPU @ 2.60GHz with six cores.

The code itself was written in Python in a Jupyter Notebook. Numerous libraries were used. The most important of these were:

- PyTorch[2]
- scikit-learn[3]
- pandas[4]
- langdetect[5]

The BERT model itself was sourced from Huggingface[6].

6.1.2 Implementation

The following section goes through and explains one implementation in more detail.

Doka GmbH provided the data as JSON lines. Therefore, this was first loaded into a Pandas dataframe.

Listing 6.1: Tokenizer

```
# Tokenizer
PRETRAINED_LM = "bert-base-multilingual-cased"
tokenizer = BertTokenizer.from_pretrained(PRETRAINED_LM)
```

The first step was to download the tokenizer (Listing 6.1) for the "bert-base-multilingual-cased" model using the transformer library. This model was chosen because it supports multiple languages. The case-sensitive model was selected. Preliminary investigations have shown that this model performs slightly better on this data set than a non-case-sensitive model.

The next step is to introduce a function (Listing 6.2) that processes the text so that the BERT model can also process the text later. For example, the [CLS] token is inserted at the beginning and the [SEP] token at the end of the document. It finally returns the 'input_ids' and the 'attention_masks'. Both training and test data pass through this function.

[2]https://pytorch.org/
[3]https://scikit-learn.org/stable/
[4]https://pandas.pydata.org/
[5]https://pypi.org/project/langdetect/
[6]https://huggingface.co/

Listing 6.2: Encoding

```
def encode(docs):
    encoded_dict = tokenizer.batch_encode_plus(docs,
            add_special_tokens=True,
            max_length=128, padding='max_length',
            return_attention_mask=True,
            truncation=True, return_tensors='pt')
    input_ids = encoded_dict['input_ids']
    attention_masks = encoded_dict['attention_mask']
    return input_ids, attention_masks

#Encode Training and Test prompts
train_input_ids, train_att_masks =
encode(df_sl_training['prompt'].values.tolist())
test_input_ids, test_att_masks =
encode(df_sl_test['prompt'].values.tolist())
```

The 'completion' column contains the target variables for training and testing. These are converted into tensor objects (Listing 6.3) so that they can be used later. Torch tensors are multi-dimensional arrays used in PyTorch to store data and perform operations on it.

Listing 6.3: Convert labels to torch tensors

```
# Convert labels to torch tensors
train_y =
   torch.LongTensor(df_sl_training['completion'].values.tolist())
test_y = torch.LongTensor(df_sl_test['completion'].values.tolist())
```

A DataLoader (Listing 6.4) is then created, which organizes the data in batches. These batches are used for the training process. The 'BATCH_SIZE' is defined as 16, as recommended in the work described above.

Listing 6.4: Dataloader

```
# Dataloader
BATCH_SIZE = 16
train_dataset = TensorDataset(train_input_ids, train_att_masks,
   train_y)
train_sampler = RandomSampler(train_dataset)
train_dataloader = DataLoader(train_dataset,
   sampler=train_sampler, batch_size=BATCH_SIZE)
```

It is time to initialize the model (Listing 6.5) itself. The BERT model is initialized and configured for sequence classification in order to adapt it to the specific requirements of the training data set, including the number of output classes.

Listing 6.5: Initialize Model

```
#Initialize Model
N_labels = max(df_sl_training['completion']) + 1
model = BertForSequenceClassification.from_pretrained
(PRETRAINED_LM,
 num_labels=N_labels,
 output_attentions=False,
 output_hidden_states=False)
```

Some parameters (Listing 6.6) are defined in this section. Each model is trained in four epochs. The learning rate is set to 2×10^{-5}. On the one hand, this corresponds to the recommendations found in the literature search. On the other hand, it is also the result of the knowledge gained by trying out different parameters. It has been shown that the learning rate should be as low as possible with such a large amount of training data. Furthermore, it was found that fewer or more epochs worsen the result. An Adam Optimiser is also used to adjust the learning rate linearly. The Adam Optimiser adjusts the learning rates for each parameter according to the speed and direction of its update. This results in faster training and more robust results.

Listing 6.6: Set parameters

```
#Parameters
EPOCHS = 4
LEARNING_RATE = 2e-5

optimizer = AdamW(model.parameters(), lr=LEARNING_RATE)
scheduler = get_linear_schedule_with_warmup(optimizer,
   num_warmup_steps=0,
   num_training_steps=len(train_dataloader)*EPOCHS)
```

Now that all parameters have been set and the models and tokenizer have been loaded and initialized, fine-tuning can begin. To do this, the model was called and trained. Finally, the model was saved so that it could be reused anytime. The training time was relatively long because, as already mentioned, the CPU was used for training. Training took around ten hours for each epoch.

This procedure was repeated for each of the other four target variables, resulting in five models, which are tested and evaluated in the next chapter.

6.2 Fine-tuning of GPT

This section shows how to fine-tune the GPT model. This requires relatively little prior knowledge as the model is trained using an API request. Doka GmbH made the API request itself. However, the evaluation was carried out by the author of this thesis. Five models were generated with GPT, one for each target variable.

Unlike BERT, an open-source model, GPT is provided by OpenAI, and you have to pay to use the models. Four models could be fine-tuned when the model was fine-tuned (October 2023). They were Ada, Babbge, Curie, and Davinci. OpenAI recommended fine-tuning the Babbage model for the text classification task. This recommendation was verified and confirmed by small pre-tests. Currently (as of March 2024), only three models can be fine-tuned. These are Babbage, Davinci, and GPT 3.5 Turbo. OpenAI now recommends the GPT 3.5 Turbo model for tuning. However, this was not available at the time of the training. The cost of fine-tuning is \$0.0004 / 1k tokens as seen in Table 6.2. OpenAI then charges \$0.0016 / 1k tokens for use.

Table 6.2: Costs per GPT model

Modell	**Training**	**Input usage**	**Output usage**
gpt-3.5-turbo	\$0.0080 / 1K tokens	\$0.0030 / 1K tokens	\$0.0060 / 1K tokens
davinci-002	\$0.0060 / 1K tokens	\$0.0120 / 1K tokens	\$0.0120 / 1K tokens
babbage-002	\$0.0004 / 1K tokens	\$0.0016 / 1K tokens	\$0.0016 / 1K tokens

Training the model is pretty straightforward, as all you need to do is identify yourself using an API key, then specify the model you want to use and submit the training data file (LISTING 6.7).

Listing 6.7: Fine-tuning of GPT Babbage

```python
from openai import OpenAI
client = OpenAI()
openai.api_key = 'XXXXXXXXXX'

client.fine_tuning.jobs.create(
  training_file="PRIO_10_10.jsonl",
  model="gpt-3.5-babbage-002"
)
```

After a few hours, the model is fully trained and can be used and evaluated. Unfortunately, there are limited options for setting the parameters. OpenAI itself sets these. You can retrieve them afterwards using an API request (Listing 6.8). It turns out that the Babbage model was also trained in 4 epochs like BERT. The batch size is 128, and an ID identifies the model. This ID can then invoke the model and send the tickets for classification.

Listing 6.8: Getting parameters

```
openai.FineTune.retrieve(id = XXXXXXXXXXX)

#RESPONSE
<FineTune fine-tune id=ft-XXXXXXXXXXX at 0x2394e363830> JSON: {
  "object": "fine-tune",
  "id": "ft-XXXXXXXXXXXX",
  "hyperparams": {
  "n_epochs": 4,
  "batch_size": 128,
  "prompt_loss_weight": 0.01,
  "classification_n_classes": 75,
  "learning_rate_multiplier": 0.1,
  "compute_classification_metrics": true  }
```

7

Evaluation

Now that the models have been fine-tuned to the Doka Gmbh tickets, the performance of the two models can finally be evaluated and compared. For the evaluation, 1.000 tickets not yet seen by the model were randomly selected from the test data set. The same tickets were then used for the BERT and GPT models. Only 1.000 tickets were used as a test data set to keep costs low. However, larger data sets were used for testing purposes to determine whether there were any differences. These showed almost the same results as the test data sets used in this thesis. Firstly, it is shown how to call up the models to obtain a classification for a ticket. An in-depth evaluation of each target variable follows this. The chapter concludes with an overall view of the results and a judgment of the results.

7.1 Classify unseen ticket

This shows how a classification works for a ticket that has not yet been seen. Custom methods have been programmed to send the entire data set to the model at once to classify the entire test set.

7.1.1 Usage of BERT

If one wants a classification from BERT for a ticket, it is first necessary to load the fine-tuned model and the tokenizer (Listing 7.1). The text must then be preprocessed so that BERT can process it.

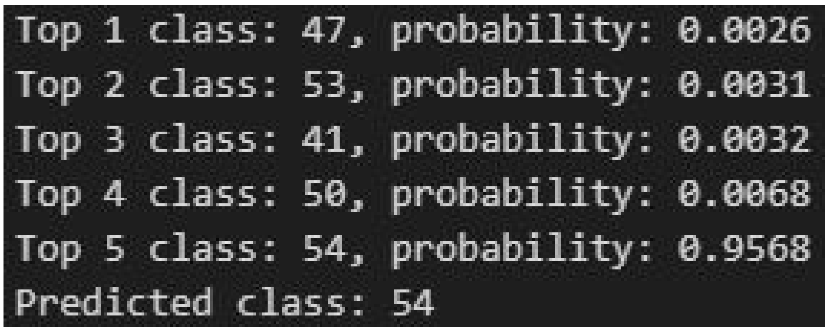
Listing 7.1: Request classification for one ticket BERT

```
#Path to fine-tuned model
model_path = r'XXXXXXX\final_bert_sl_aeda_model'
#Load tokenizer and model
tokenizer =
   BertTokenizer.from_pretrained("bert-base-multilingual-cased")
model = BertForSequenceClassification.from_pretrained(model_path)

```

```
#ticket text
text = "Subject: Email Issue: Not Receiving Emails\nContent:
    Hello, I currently face some issues with my Outlook.
A colleague just sent me an e-mail but they  d o n t  appear in my
    mailbox. Could you please check if
everything is fine with my mail address user@website.com? Best
    regards, John Doe\n"
```

The result is then either the predicted class or, as can be seen in Fig. 7.1, it is also possible to obtain the TOP5 classes with the respective probabilities.



```
Top 1 class: 47, probability: 0.0026
Top 2 class: 53, probability: 0.0031
Top 3 class: 41, probability: 0.0032
Top 4 class: 50, probability: 0.0068
Top 5 class: 54, probability: 0.9568
Predicted class: 54
```

Figure 7.1: Response from BERT

7.1.2 Usage of GPT

To get the classification for a ticket, OpenAI again requires an API request, as seen in LISTING 7.2. The ticket text is passed. In addition, some parameters can be passed. temperature=0 controls the predictability of the answers. This means that the model will give the most likely prediction. For classification tasks, it is essential that this value is zero; for the text generation task, this value can be changed to get more creative answers. logprobs=5 indicates that the five most probable results should be displayed. Only the most probable result is used in this paper.

Listing 7.2: Request classification for one ticket GPT

```
openai.Completion.create(model=ft_model, prompt=
'Subject: Email Issue: Not Receiving Emails\nContent:
Hello, I currently face some issues with my Outlook.
A colleague just sent me an e-mail but they
 d o n t  appear in my mailbox. Could you please check if everything
is fine with my mail ad-dress user@website.com?
Best regards, John Doe \n\n
###\n\n', temperature=0, stop="\n\n###\n\n", logprobs=5,
    max_tokens=1)
```

The response is a JSON object (LISTING 7.3) containing all the necessary information. The five most likely classes are specified, with the most likely being highlighted. The probabilities are given as logarithms of the probabilities. For example, if you want the probability in a percentage, you can use the exponential function (e^x). In the case shown here, the probability that the ticket belongs to class 50 would be 71.82% and is therefore the most probable class.

Listing 7.3: GPT Probabilities

```
<OpenAIObject text_completion id=cmpl-XXXXXXXXXX at 0x28e7ceb3880>
   JSON: {
"id": "cmpl-XXXXXXXXXX",
"object": "text_completion",
"created": 1699020659,
"model": "babbage:ft-doka-2023-10-05-18-30-10",
"choices": [
    {
        "text": " 50",
        "index": 0,
        "logprobs":
 {  "tokens": [
 " 50"
 ],
 "token_logprobs": [  -0.33095223
 ],
 "top_logprobs": [
 {
 " 30": -3.8044028,
 " 50": -0.33095223,
 " 36": -2.6513915,
 " 6": -3.7689948,
 " 67": -2.7258105  }  ],
...
    "prompt_tokens": 66,
    "completion_tokens": 1,
    "total_tokens": 67
    }
}
```

7.2 Evaluation

Now, it's time to examine the results more closely. This section will discuss each attribute, followed by an overall assessment.

The metrics used were the standard metrics for classifiers, including Accuracy, Precision, Recall, and F1-Score. The weighted metrics were used here in each case. Accuracy is used as the main comparison metric. The metrics used are briefly explained below:

Accuracy
Definition: Accuracy is the ratio of correctly predicted instances to the total instances.
Formula:

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

Precision
Definition: Precision is the ratio of correctly predicted positive observations to the total predicted positives.
Formula:

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

Recall
Definition: Recall is the ratio of correctly predicted positive observations to all the observations in the actual class.
Formula:

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

F1 Score
Definition: The F1 Score is the harmonic mean of Precision and Recall.
Formula:

$$F1\,Score = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$$

7.2.1 Form

The first models to be evaluated (Table 7.1) for performance are those for the target variable Form. About 80% of the tickets have no form at all; the remaining 20% are divided into 99 different classes. The fine-tuned GPT 3.5 model was able to achieve an accuracy of 93.5% for this variable. Precision, recall, and F1 scores are also within this range and show no noteworthy deviations. In comparison, the fine-tuned BERT model achieved an accuracy of 91.2%. This is approximately 2.3 percentage points behind the performance of the GPT model. The other metrics for the BERT model are 87.1% for precision, 91.2% for recall, and 88.9% for F1 score. Therefore, the GPT model performs slightly better than BERT for this attribute.

If we only take a look at the "no form" class, we can see that GPT recognizes it correctly in 97.3% of cases. BERT, on the other hand, has a slightly better value here, acknowledging that there is no form in 98.7% of cases. There are many training tickets for this class, as it accounts for 80% of cases. Therefore, it can be concluded that GPT performs better than BERT in cases with less training data.

Table 7.1: Evaluation Form

Metric	**GPT 3.5**	**BERT**
Accuracy	**93,5** %	**91,2** %
Precision	92,5 %	87,1 %
Recall	93,5 %	91,2 %
F1-Score	92,8 %	88,9 %

7.2.2 Priority

The second target variable that is evaluated is the Priority variable. It indicates how urgent a ticket is, and there are only four different classes. The fine-tuned GPT model achieves an accuracy of 97.5% (Table 7.2) on the test dataset. The other metrics show no particular deviation from accuracy. The BERT model achieves an accuracy of 98.6% for this attribute. This is 1.3 percentage points higher than the GPT model. It can, therefore, be concluded that BERT performs better for this objective.

Table 7.2: Evaluation Priority

Metric	**GPT 3.5**	**BERT**
Accuracy	**97,5** %	**98,6** %
Precision	97,5 %	96,7 %
Recall	97,5 %	98,1 %
F1-Score	97,1 %	98,8 %

As you can see from the heatmap (Fig. 7.2) for the fine-tuned GPT model, the model has a hard time especially with tickets with the priority "High". These are incorrectly assigned to the normal category in 90% of cases.

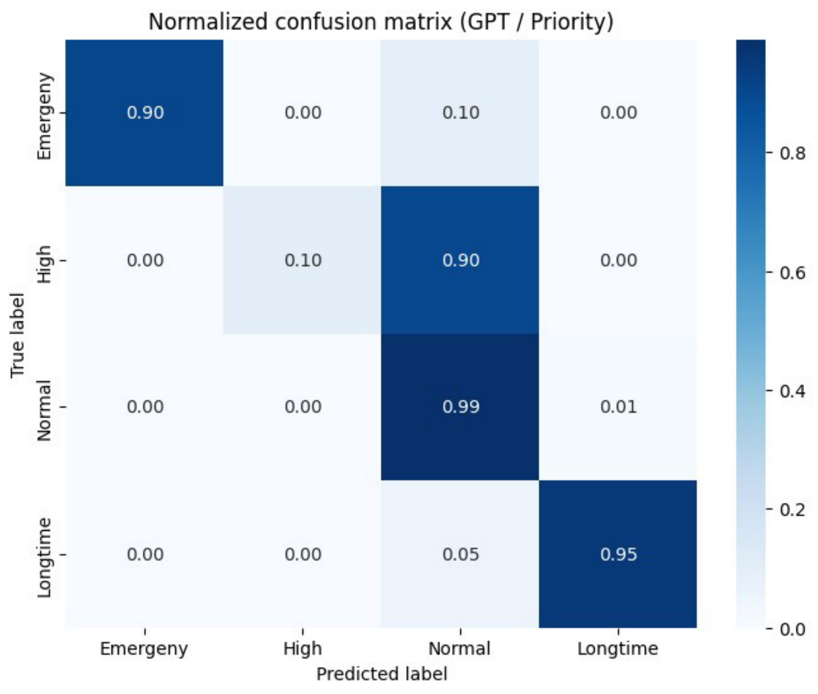


Figure 7.2: Normalized confusion matrix (GPT / Priority)

7.2.3 Queuename

The target variable Queuename consists of a total of six classes to which tickets can belong. The GPT model (Table 7.3) achieved an accuracy of 88.6% for this variable. The fine-tuned BERT model achieved a similar accuracy, namely 88.4%. This is, therefore, only 0.2 percentage points behind the performance of the GPT model. The other metrics do not indicate any anomaly in the models.

Table 7.3: Evaluation Queuename

Metric	**GPT 3.5**	**BERT**
Accuracy	**88,6** %	**88,4** %
Precision	88,3 %	88,1 %
Recall	88,6 %	88,4 %
F1-Score	88,4 %	88,2 %

As one can see from the confusion matrix for the variable Queuename (Fig. 7.3), both models perform almost equally well. Especially for the Incident and Ittask classes both have their problems, with BERT performing slightly worse, especially for the Ittask. BERT is also usually just behind GPT for the other classes, which means that GPT is 0.2 percentage points better in terms of overall performance.

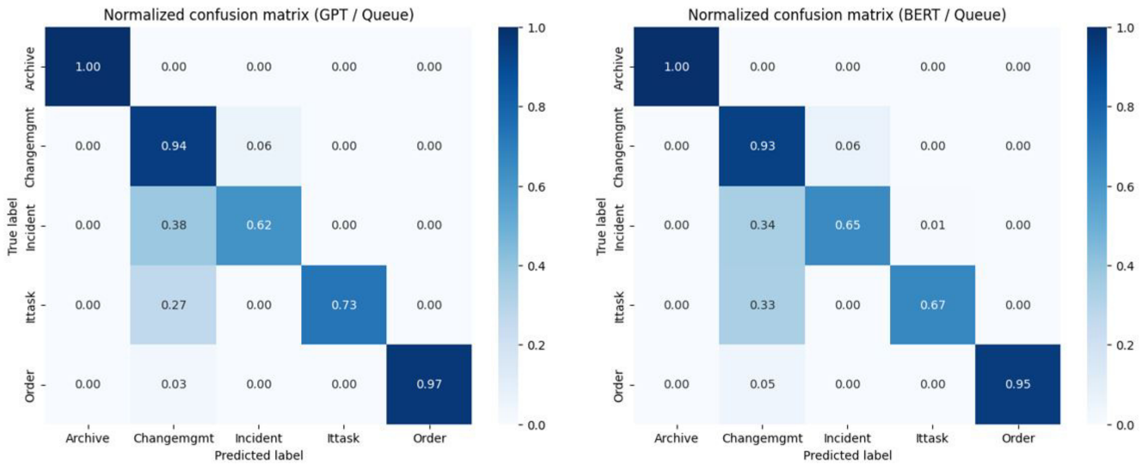


Figure 7.3: Comparison of GPT and BERT / Heatmap for Queuename

7.2.4 Servicetype

It is now time to examine the most difficult predictable target variable, namely Servicetype. A ticket can belong to one of 116 different classes, which are quite balanced. The GPT model achieves an accuracy of 81.3% (Table 7.4) under these challenging conditions. The precision is 80.3%, the recall is 81.3%, and the F1 score is 80.4%. In comparison, the BERT model shows a much weaker performance. The accuracy is only 72.6%. This is 8.7 percentage points less accurate than the fine-tuned GPT model. The precision is 67.9%, the recall is 72.6%, and the F1 score is 69.3%. It has already been shown that BERT has difficulties with too many classes and, correspondingly, less training data. It can be seen that the GPT model copes much better with this scenario.

Table 7.4: Evaluation Servicetype

Metric	**GPT 3.5**	**BERT**
Accuracy	**81,3** %	**72,6** %
Precision	80,3 %	67,9 %
Recall	81,3 %	72,6 %
F1-Score	80,4 %	69,3 %

For better comprehensibility, only a subtest set of 1.000 unseen tickets was used for the visualization, and only classes with more than ten tickets were considered. In Fig. 7.4 the results are shown for GPT, in Fig. 7.5 for BERT. This visualization clearly shows how BERT achieves worse values in terms of accuracy than the fine-tuned GPT model across all classes.

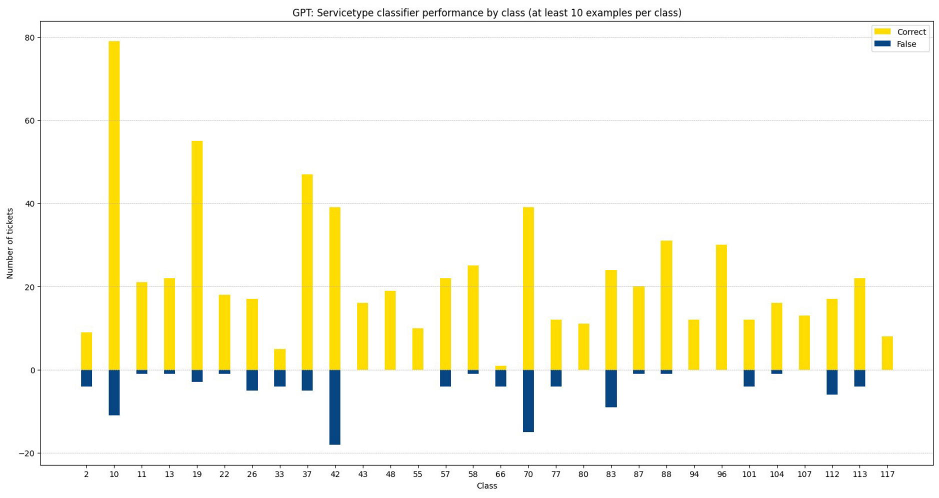


Figure 7.4: GPT: Servicetype classifier performance by class (at least 10 examples per class)

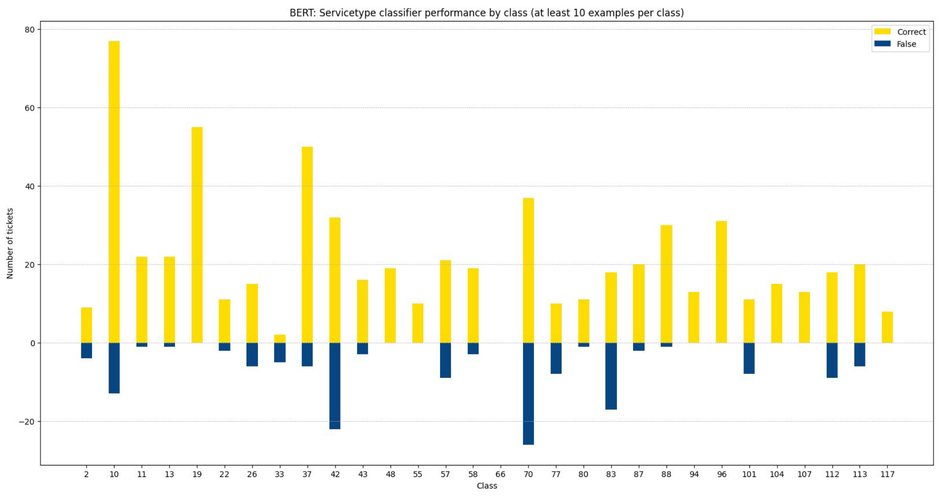


Figure 7.5: BERT: Servicetype classifier performance by class (at least 10 examples per class)

7.2.5 Serviceline

The Serviceline attribute is considered the most essential attribute for a ticket, as it indicates who is supposed to process the ticket. There are a total of 72 Servicelines, each of which can be responsible for the ticket. As explained in the chapter on data understanding, around half of the tickets are processed directly by the helpdesk and, therefore, the IT-HD Serviceline. The fine-tuned GPT model achieves an accuracy of 80.5% (Table 7.5) for this attribute. In contrast, the accuracy of the BERT model is 77.0%. This also shows that the GPT model can handle the relatively large number of classes better than the fine-tuned BERT model. The performance is 3.5 percentage points higher than the fine-tuned BERT model.

Table 7.5: Evaluation Serviceline

Metric	GPT 3.5	BERT
Accuracy	**80,5** %	**77,0** %
Precision	80,9 %	76,6 %
Recall	80,5 %	77,0 %
F1-Score	80,3 %	75,5 %

If we only consider the IT-HD class in this case, which, as already mentioned, accounts for around half of the tickets, GPT has a detection rate of 83.4%. BERT recognises this class only slightly worse and has an accuracy of 82.2%. The difference here is, therefore, only 1.2 percentage points, while the difference across all classes is 3.5 percentage points. This further reinforces the impression that BERT needs more training examples than GPT.

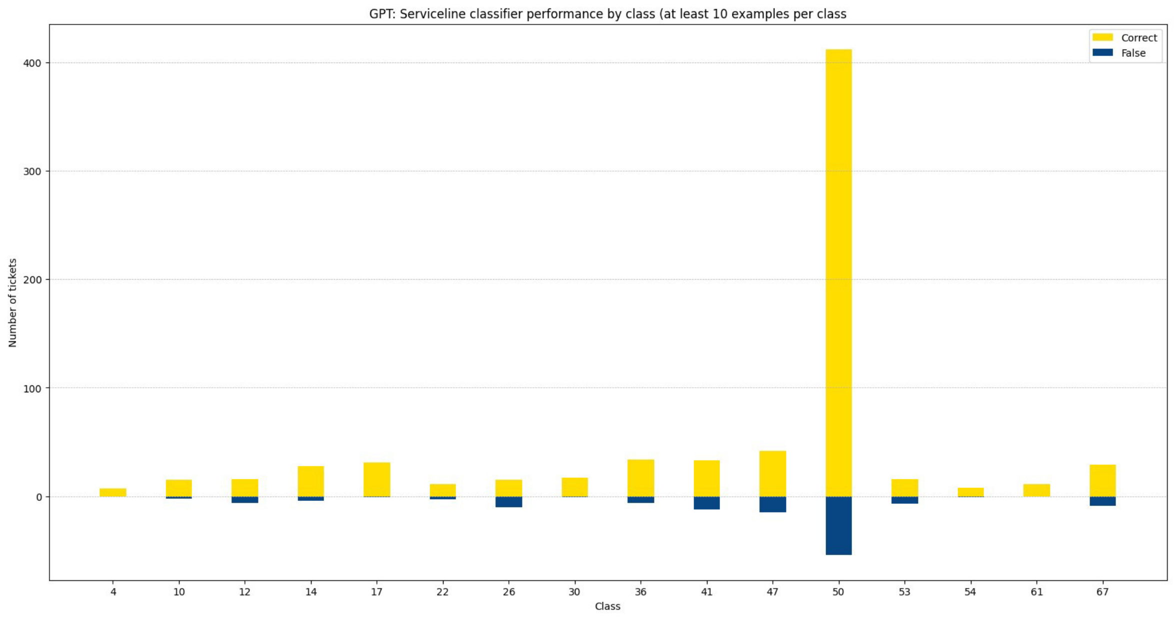


Figure 7.6: GPT: Serviceline classifier performance by class (at least 10 examples per class)

For better comprehensibility, only a subtest set of 1.000 unseen tickets was used for the visualization, and only classes with more than ten tickets were considered. In Fig. 7.6 the results are shown for GPT, in Fig. 7.7 for BERT. It can be seen that GPT performs better, especially for classes where fewer data records are available.

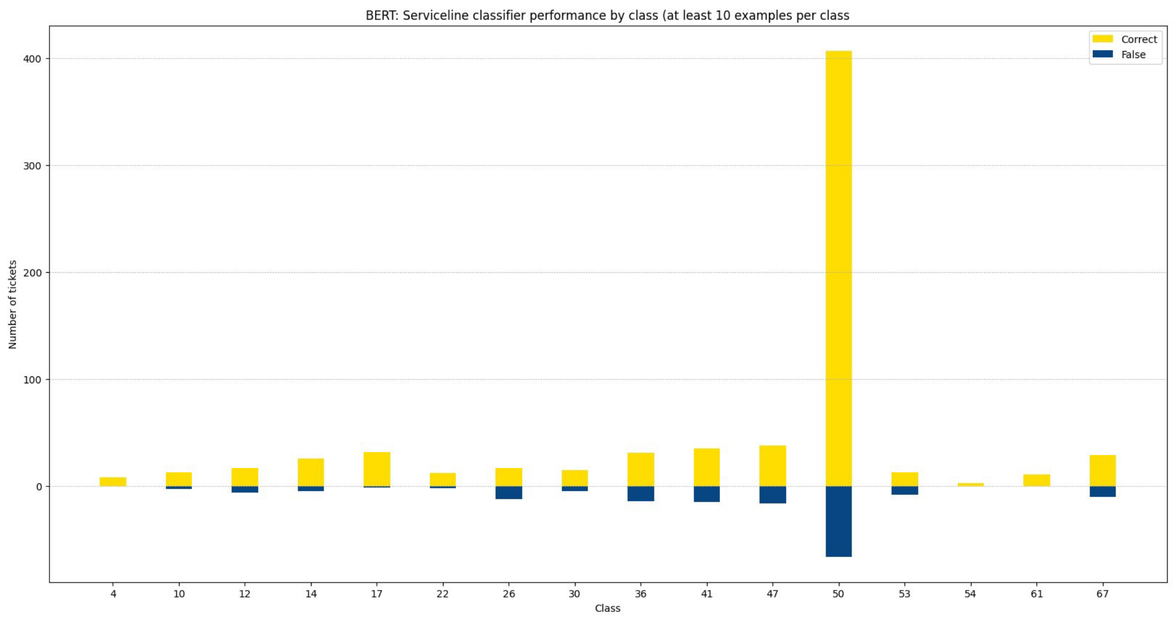


Figure 7.7: BERT: Serviceline classifier performance by class (at least 10 examples per class)

7.2.6 BERT versus GPT

Now that each target variable has been considered individually, a holistic view of the results (Table 7.6) is taken, and the results are contextualized. For the most important attribute, the Serviceline, the fine-tuned GPT model achieves a performance of 93.5%. This compares with 91.2% for the fine-tuned BERT model. For the servicetype, GPT achieves 81.3%, and the BERT model lags far behind with 72.6%. Queuename can be predicted by GPT with an accuracy of 88.6% and by BERT with an accuracy of 88.4%. For Priority, the fine-tuned BERT model performs slightly better with an accuracy of 98.6% compared to GPT, which has a value of 97.5%. The form can again be predicted better by GPT, namely with 93.5%. The fine-tuned BERT model achieves a percentage value of 91.2%.

Table 7.6: Evaluation BERT versus GPT

Accuracy	**GPT 3.5**	**BERT**
Serviceline	**80,5 %**	77 %
Servicetype	**81,3 %**	72,6 %
Queuename	**88,6 %**	88,4 %
Priority	97,5 %	**98,6 %**
Form	**93,5 %**	91,2 %

This shows (Fig. 7.8) that the fine-tuned GPT model performs better than the fine-tuned BERT model in 4 out of 5 cases.

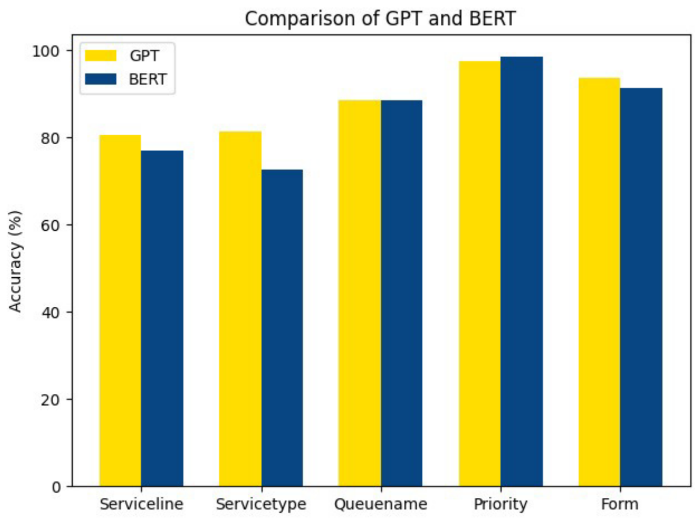


Figure 7.8: Comparison of GPT and BERT

If you take a closer look at the results, you can see that BERT performs better than GPT Fig. 7.9 for Priority, where there are only four classes. For Queuename, where there are six classes, both are about the same in terms of accuracy. With Form, for example, there are also 100 classes, but since most of them belong to one class, BERT has enough training data to perform only slightly worse than the fine-tuned GPT model. With the Serviceline, where around half of the tickets belong to one class, the difference in performance rises to 3.5 percentage points. For the Servicetype, the difference is highest at 8.7 percentage points. On the one hand, there are a huge number of classes here; on the other hand, these are more balanced than the other variables. This means that there is less training data per class everywhere.

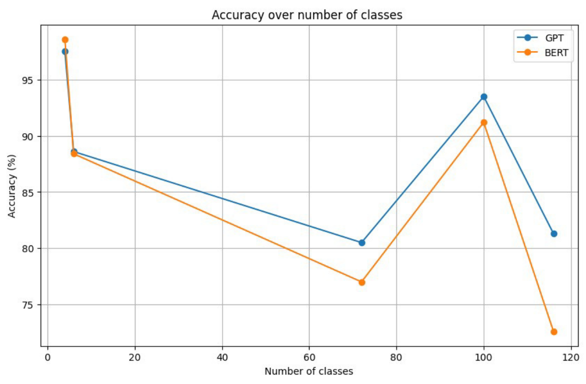


Figure 7.9: Accuracy over number of classes

The next chapter examines data augmentation to address this problem. This should help artificially generate training data. The focus is on specifically strengthening underrepresented classes and thus improving overall performance.

8

Data Augmentation

As one can see, excellent results were achieved with fine-tuning BERT and GPT for ticket classification. However, it should be noted that the two most essential target variables, Serviceline and Servicetype, performed worse than the other variables. To address this problem, DA will be used for the BERT model to evaluate whether this approach improves performance.

DA is a method used to increase the training data size by making slightly modified copies of existing data or artificially generating completely new data (B. Li et al., 2022). This is necessary because the data set must be large enough for the training so that the model can also deal well with unseen data (Karimi et al., 2021). Since it is often difficult to collect a large enough data set of good quality, DA is frequently used in the field of NLP to counter this problem (Wei & Zou, 2019).

8.1 Data Augmentation methods

Li et al. (B. Li et al., 2022) categorize the various DA methods in their paper "Data augmentation approach in natural language processing: A survey." This master thesis uses this categorization to provide a better overview.

Based on the diversity of the data generated, they divide the DA methods into three categories (Fig. 8.1): paraphrasing, noising, and sampling, which are briefly presented below.

8.1.1 Paraphrasing-based methods

The paraphrasing-based methods generate data with only a slight semantic deviation from the original data, and as little information as possible should be lost from the original data. Here, the authors differentiate between the categories of thesaurus, semantic embeddings, language models, rules, and machine translation.

The widely used Easy Data Augmentation Techniques (Wei & Zou, 2019) method, in which words are replaced by synonyms using WordNet, can represent the thesaurus approach.

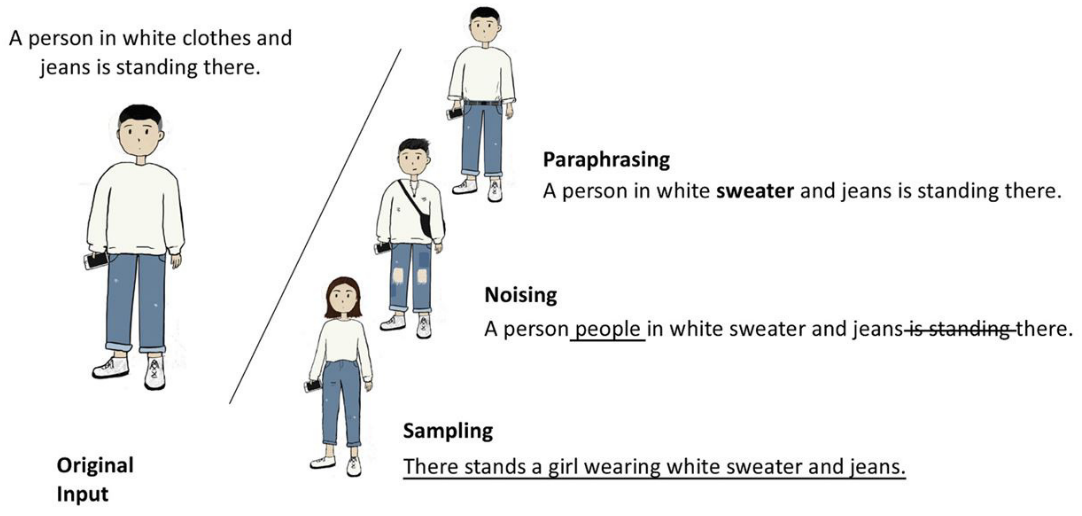

A person in white clothes and jeans is standing there.
Paraphrasing
A person in white **sweater** and jeans is standing there.
Noising
A person people in white sweater and jeans ~~is standing~~ there.
Sampling
There stands a girl wearing white sweater and jeans.
Original Input

Figure 8.1: Data augmentation categories (B. Li et al., 2022)

Back-translation is also a top-rated member of this category. This involves translating the whole sentence into another language and then translating it back again. This creates a new sentence that is different from the original without changing the meaning. English-French translation models are often used here (Fabbri et al., 2020; Xie et al., 2020).

8.1.2 Noising-based methods

Noising attempts to generate new data by adding low-level noise. The semantics are not changed. This not only increases the training data set but also increases the robustness of the model. There are several approaches to this. Swapping involves swapping words within a sentence. Deletion involves deleting a randomly selected word from the sentence. Insertion involves randomly inserting a word into the sentence. The EDA (Wei & Zou, 2019) approach also uses these three methods. This method, therefore, includes both paraphrase-based methods and noise-based methods. Another approach of the noise-based method is substitution. Here, words are replaced by any other word. For example, a list of common misspellings can be used to generate data (Regina et al., 2020) artificially. Another representative is AEDA (An Easier Data Augmentation Technique for Text Classification (Karimi et al., 2021)). Punctuation marks are randomly injected into the original sentence to generate augmented data.

8.1.3 Sampling-based methods

Sampling-based methods target the data distribution and generate new samples within it. Sampling-based methods are task-specific and require task information such as labels and data format. They, therefore, ensure validity and diversity. They are generally more challenging to implement than the methods shown above. There is a distinction between rules, non-pre-trained, pretrained, self-training, and confusion methods. In the case of rules, some approaches swap the subject and object of the original sentence and swap predicate verbs to the passive form (Min et al., 2020). Another promising approach is the usage of pre-trained models. Here, additional data generated by LLMs such as GPT-2 or BART is used as augmented data (Kumar et al., 2020).

8.2 Applied data augmentation methods

As we have seen in the previous section, there are many different approaches to generating augmented data. In this paper, two approaches are implemented to evaluate the impact on the performance of the fine-tuned models. According to the literature analysis by Li et al. (Q. Li et al., 2022), paraphrase based or sampling based methods are mostly used for data augmentation for text classification tasks. It would be obvious to use pre-trained models such as GPT to generate augmented data in this work. However, this was not considered due to business constraints and the cost involved. Of the paraphrase based models, back translation of this task would be an option. This was evaluated in more detail. However, as the data generation could only take place locally due to company policy, it quickly became apparent that the generation of a relevant number of augmented datasets would have an extremely long runtime. Small tests were carried out with Google T5 and mBART, which confirmed this impression (Kumar et al., 2020).

The first method used for DA is the Easy Data Augmentation Technique (Wei & Zou, 2019). This includes both paraphrase based methods and noise based methods. EDA is characterized by the fact that it combines different ways of generating data. These are synonymous replacement, random insertion, random swap, and random deletion. This ensures that a large variety of data is generated. The authors of the paper show that using EDA, only 50% of the training data was needed to achieve the same accuracy as the full training data. The second method used is An Easier Data Augmentation Technique for Text Classification by Karimi et al. (Karimi et al., 2021). As explained above, punctuation marks are randomly inserted into the original text. The authors show that AEDA outperforms EDA in their test setting. In this thesis, these two methods are implemented to see if they can improve the performance of the models, and they are also evaluated against each other.

8.2.1 DA for Serviceline

The first method to be implemented is EDA. The implementation follows the code published for the paper[1]. For the target variable, the distribution shows that the class "IT-HD" occurs approximately as often as all the others combined. Therefore, an attempt is made to use EDA for all other classes in order to minimize the imbalance of the dataset. First, all German and English training data was filtered out using the langdetect Python library. The result is a total of 49.924 tickets for which augmented data is to be created. The random deletion and random swap methods were applied for both, the English and German data.

[1]https://github.com/jasonwei20/eda_nlp

WordNet was used for synonym replacement and random insertion. This is a database that provides English synonyms. As no satisfactory German alternative could be found despite an intensive search, these two methods were only used for the English tickets. Although there is a German alternative, GermaNet, it could not be used due to licensing issues. As a result, four artificially generated tickets were created for each English ticket, and two artificially generated tickets were created for each German ticket. This resulted in a total of 151.642 new training tickets being created.

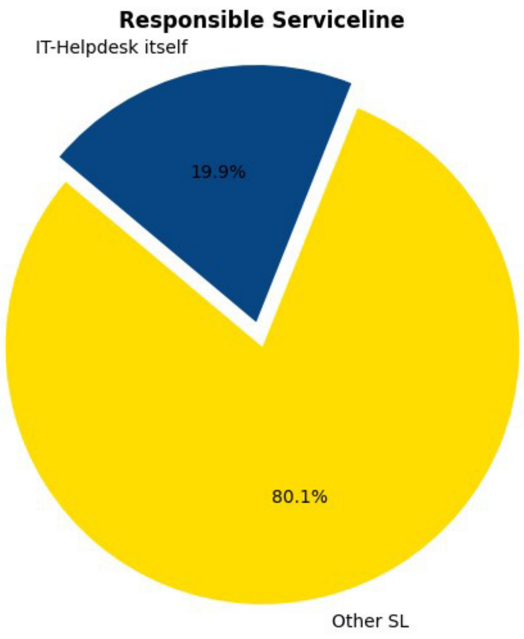


Figure 8.2: Distribution Serviceline with data augmentation

As one can see in Fig. 8.2, the use of data augmentation makes it possible to change the proportion of tickets that are handled directly by the helpdesk. Initially, just over half were allocated to the IT-HD serviceline. Now the ratio in the training data has changed dramatically and only 19.9% point to the IT-HD class. 80.1% are handled by other service lines.

AEDA is implemented as the second method. The implementation is again based on the code published with the paper[2]. One of the punctuation marks {".", ";", "?", ":", "!", ","} is randomly inserted at a randomly selected position. In order to create a comparable basis, AEDA also created four artificially generated tickets for the English tickets and only two artificial tickets for the German tickets.

The models were then completely retrained with the original data set and the augmented data set.

[2]https://github.com/akkarimi/aeda_nlp

8.2.2 DA for Servicetype

When implementing EDA and AEDA for the target variable Servicetype, the procedure was exactly the same as for the target variable Serviceline. As the class ratio for Servicetype is much more balanced, only classes that occur less than 2.000 times were taken into account. In this way, 86.276 new english tickets and 38.882 new german tickets were created. This is a compromise to ensure a reasonable runtime and to generate enough data sets to improve the model.

The models were them completely retrained with the original data set and the augmented data set.

8.2.3 Example

Table 8.1 shows an example of how EDA and AEDA change the content of a ticket and thus ensure more diversity. In the synonym replacement, some words have been replaced by synonyms. For example, "issue" has been replaced by "problem". With random insertion, words are added, in this case "face some issues" became "face some unexpected issues". Random swap simply swapped two words, in this case "inbox" and "e-mail". Random deletion deletes a word at random. So "Best regards" simply became "Regards". With AEDA, punctuation is inserted at random. Here in the middle of the sentence.

Table 8.1: Ticket example EDA and AEDA

Original	Subject: Email Issue: Not Receiving Emails\nContent: Hello, I currently face some issues with my Outlook. A colleague just sent me an e-mail, but they don't appear in my mailbox. Could you please check if everything is fine with my email address, philipp.engelbrechtsmueller@doka.com? Best regards, Philipp Engelbrechtsmüller\n
SR (Synonym Replacement)	Subject: Email **Problem**: Not Receiving Messages\nContent: **Greetings**, I am currently **encountering** some problems with my Outlook. A colleague has **recently** sent me an e-mail but they are not showing up in my mailbox. Could you please **verify** if everything is okay with my email address user@website.com? **Warm** regards, John Doe\n
RI (Random Insertion)	Subject: Email Issue: Not Receiving Emails\nContent: Hello, I currently face some **unexpected** issues with my Outlook. A colleague just sent me an e-mail but **sadly** they don't appear in my mailbox. Could you please check if everything is fine with my **personal** mail address user@website.com? Best regards, John Doe\n
RS (Random Swap)	Subject: Email Issue: Not Receiving Emails\nContent: Hello, I currently face some issues with my Outlook. A colleague just sent me an **inbox** but they don't appear in my **e-mail**. Could you please check if everything is fine with my email address user@website.com? Best regards, John Doe\n
RD (Random Deletion)	Subject: Email Issue: Not Receiving Emails\nContent: Hello, I face some issues with my Outlook. A colleague sent me an e-mail but they don't appear in my mailbox. Could you please check if everything is fine with my mail address user@website.com? Regards, John Doe\n
AEDA	Subject: Email Issue: Not Receiving Emails\nContent: Hello, I currently face some issues with my Outlook. A colleague **;** just sent me an e-mail, but they don't appear in my mailbox. Could you please check if everything is fine with my mail address user@website.com? Best regards, John Doe\n

8.3 Evaluation

This section discusses the results of the models that were enriched with augmented data. Firstly, the effects of using EDA and AEDA in the Servicetype model are shown, followed by an evaluation of the Serviceline model.

8.3.1 Results DA for Servicetype

The original BERT model for the Servicetype attribute was fine-tuned with 101.938 training tickets. After the data augmentation process, 227.096 tickets were available. The entire model was retrained entirely with more than double the number of tickets. As can be seen in Table 8.2, the original model achieved an accuracy of 72.60% rounded. By using EDA to generate additional training examples, a notable increase in accuracy was achieved to 76.70%. This is an increase of 4.4 percentage points. This is a massive increase over the original model. Using AEDA increased the accuracy to 77.50%. This is an increase of 4.9 percentage points.

Table 8.2: Servicetype model evaluation

Servicetype	Accuracy
Original	72,60 %
+EDA	76,70 %
+AEDA	77,50 %

Although GPT's accuracy of 81.30% could not be surpassed, it has been shown that data augmentation techniques can improve performance. A comparison of the performance of EDA and AEDA shows that AEDA performs 0.8 percentage points better (Fig. 8.3). This is consistent with the results of the literature review.

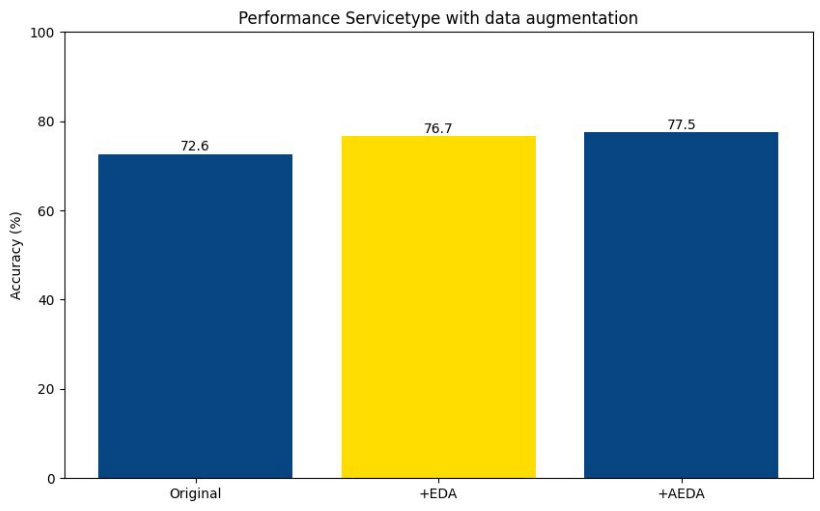


Figure 8.3: Performance Servicetype with data augmentation

8.3.2 Results DA for Serviceline

Next, we look at the performance of the Serviceline model (Table 8.3) using data augmentation. The original model was trained on 102.039 tickets. The data augmentation process increased this number to 253.681 tickets. The main objective was to correct the dataset imbalance, as more than half of the tickets were in the IT-HD class. The original model achieved an accuracy of about 77%. The model enriched with EDA data increased this accuracy to 79.80%. This is an increase of 2.8 percentage points. Generating data from AEDA increased the accuracy to 79.90%. This is an increase of 2.9 percentage points.

Table 8.3: Serviceline model evaluation

Serviceline	Accuracy
Original	77,00 %
+EDA	79,80 %
+AEDA	79,90 %

Overall, the increase is slightly less than in the Servicetype attribute. However, it should be noted that the Serviceline generally started at a higher level. Here, the accuracy of the original model was already 77% (Fig. 8.4), while it was only 72.6% for the Servicetype.

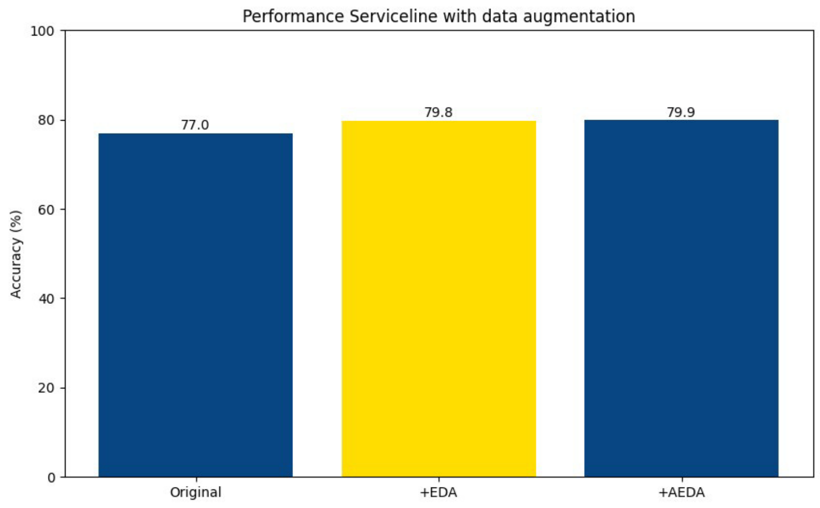


Figure 8.4: Performance Serviceline with data augmentation

8.3.3 Interpretation of the results

It was shown that both EDA and AEDA can improve the performance of the fine-tuned models. The overall increase was in line with what was expected from the literature. In particular, a notable increase was achieved for the Servicetype attribute. This is mainly because the performance of BERT decreases sharply as the number of classes increases. As already shown, GPT can handle this better when less training data is available for the respective classes. With the help of the data augmentation for BERT, this problem could be reduced somewhat. The number of classes in the Serviceline is not as high, so the performance gain from EDA and AEDA is not as high. In general, the use of DA is a balancing act. Although as much additional training data as possible should be generated, there is always a risk that the artificially generated data will become increasingly distant from the original and no longer add value to the model.

9

Conclusion

In this chapter, the thesis results are summarised again, especially concerning the two research questions in the introduction. More specifically, the subchapter 9.1 revisits the research questions and provides a final result and conclusion of the findings. 9.2 discusses the limitations of this work. The thesis ends with 9.3, which looks to the future. The aim is to show how the limitations can be overcome and what possibilities may arise due to technological progress.

9.1 Summary

The approaches available for automatic text classification were shown at the beginning of the thesis. Doka GmbH has already attempted to classify tickets using traditional classification methods. This approach was discarded due to the time-consuming feature extraction and moderate performance. Next, the deep neural networks were presented, and how they tackle the problem of ticket classification. Although they offer advantages over traditional classifiers, the literature review showed that they lag behind large language models regarding accuracy. It was found that models such as BERT and GPT are currently state-of-the-art for performing the task of text classification.

The aim of this thesis was primarily to answer the two research questions:

RQ1: How does a fine-tuned BERT model perform in the text classification task of Doka GmbH Helpdesk tickets compared to a fine-tuned GPT-3.5 model?

As the literature review shows, there are already many studies on how to fine-tune BERT for the task of text classification. However, this is still incomplete for GPT, and there are no concrete implementations of GPT fine-tuning for text classification. This may be because the models are still relatively new; they cost something, and GPT is promoted as a few-shot learner. However, preliminary work has shown that the few-shot approach does not work well for many classes. Furthermore, there are no comparisons between the performance of BERT and GPT when the fine-tuning approach is applied to both. Therefore, a GPT 3.5 Babbage model was fine-tuned in collaboration with Doka GmbH. In addition, a BERT model was also fine-tuned to predict the five target variables. This collaboration results in a fine-tuned GPT model that can be accessed via an API to classify tickets. Five BERT models have also been developed, which can be used by Doka GmbH to classify the tickets.

Evaluation of the models has shown that GPT generally performs better than BERT. In 4 out of 5 cases, better results can be achieved in accuracy. Only for the target variable Priority was it possible for BERT to beat GPT 3.5. The reason was the number of classes and the amount of training data. BERT benefits more from more training data and fewer classes. The GPT model, on the other hand, generalizes better and can classify with less training data even when there are many different target classes. This suggests that the "few-shot learner" feature of GPT also plays a role in fine-tuning.

RQ2: How do data augmentation strategies impact the performance of a fine-tuned BERT model in predicting the target variables "Serviceline" and "Servicetype"?

Since the first research question showed that BERT has problems with too little training data, the second research question was to answer whether it is possible to increase the model's performance by artificially generating additional training data. This was done for the two variables, Servicetype and Serviceline. These were chosen because they are of the highest relevance to Doka GmbH. Various approaches to data augmentation were evaluated during the literature search. Ultimately, the decision was made in favor of EDA and AEDA. The literature has already shown that these approaches can increase performance. The research question was whether these increases can be applied to other texts. For the Servicetype attribute in particular, a increase in performance was achieved by using DA. It was increased from 72.6% in the original model to 76.7% with EDA and to 77.5% with AEDA. For the Serviceline, the accuracy of the original model, which was 77.0%, was increased by 2.8 percentage points using EDA and by 2.9 percentage points using AEDA.

9.2 Limitations

About limitations, it should be mentioned that many of the Doka GmbH tickets contain screenshots. These often show the program and could increase the performance, especially regarding the target variable Servicetype. With GPT 3.5, it is not yet possible to use images and screenshots, but this would be possible with GPT 4, which was released while this paper was written.

Another limitation is that no suitable alternative to Wordnet has been found for the German tickets. Wordnet is only available for the English language; therefore, it was not possible to apply the EDA approach fully to all training tickets within the scope of this work.

Finally, it should be mentioned that there are many different variants of BERT, which, in some cases, promise better performance or aim at a smaller and more efficient model size. Due to technical limitations caused by BERT's local training, it was not possible to evaluate a large number of different variants. In addition, an effort was made to keep the starting points for BERT and GPT as similar as possible to ensure a fair comparison.

9.3 Future Work

Considering the results and limitations of the research, several areas for future research can be identified. First of all, the use of GPT 4, which, as already mentioned, allows the use of the screenshots available in the tickets. If each screenshot were labeled, it would probably be possible to improve the performance of each model. In general, only the two models, GPT from OpenAI and BERT from Google have been considered in this work. However, these are not the only LLMs currently available. For example, this comparison could also include Meta's Llama or Google's PaLM. Furthermore, GPT allows working with embeddings to understand domain-specific words better. This approach with embeddings could also be used to classify tickets. About BERT, if there are no technical limitations, it would be possible to try out different variants to evaluate whether there are performance differences in the context of ticket classification.

A possible research direction regarding the second research question would be the use of a German Wordnet alternative. This could further improve the EDA approach, as it would likely perform synonym replacement and random insertion for the German tickets as well. Testing other DA techniques for the second research question would also be possible. For example, it would be interesting to see how the computationally intensive back-translation method affects the models' performance.

Bibliography

Albawi, S., Bayat, O., Al-Azawi, S., & Ucan, O. N. (2018). Social touch gesture recognition using convolutional neural network. *Computational Intelligence and Neuroscience*, *2018*, 6973103. https://doi.org/10.1155/2018/6973103

Baeza-Yates, R., Ribeiro-Neto, B., et al. (1999). *Modern information retrieval* (Vol. 463). ACM press New York.

Chen, T., & Guestrin, C. (2016). Xgboost: A scalable tree boosting system. *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 785–794.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Diao, Y., Jamjoom, H., & Loewenstern, D. (2009). Rule-based problem classification in it service management. *2009 IEEE International Conference on Cloud Computing*, 221–228.

Doka. (2024). Doka. Retrieved March 31, 2024, from https://www.doka.com/at/index

Fabbri, A. R., Han, S., Li, H., Li, H., Ghazvininejad, M., Joty, S., Radev, D., & Mehdad, Y. (2020). Improving zero and few-shot abstractive summarization with intermediate fine-tuning and data augmentation. *arXiv preprint arXiv:2010.12836*.

Hadi, W., Al-Radaideh, Q. A., & Alhawari, S. (2018). Integrating associative rule-based classification with naıve bayes for text classification. *Applied Soft Computing*, *69*, 344–356.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735–1780.

ITIL. (2024). Itil. Retrieved March 31, 2024, from https://www.axelos.com/resource-hub/practice/readers-manual-itil-4-practice-guide

Jiao, Q. (2023). A brief survey of text classification methods. *2023 IEEE 3rd International Conference on Information Technology, Big Data and Artificial Intelligence (ICIBA)*, *3*, 1384–1389.

Karimi, A., Rossi, L., & Prati, A. (2021). Aeda: An easier data augmentation technique for text classification. *arXiv preprint arXiv:2108.13230*.

Kim, Y. (2014, October). Convolutional neural networks for sentence classification. In A. Moschitti, B. Pang & W. Daelemans (Eds.), *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)* (pp. 1746–1751). Association for Computational Linguistics. https://doi.org/10.3115/v1/D14-1181

Kumar, V., Choudhary, A., & Cho, E. (2020). Data augmentation using pre-trained transformer models. *Proceedings of the 2nd Workshop on Life-long Learning for Spoken Language Systems*, 18–26.

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

Lee, J.-S., & Hsiang, J. (2020). Patent claim generation by fine-tuning openai gpt-2. *World Patent Information*, *62*, 101983.

Li, B., Hou, Y., & Che, W. (2022). Data augmentation approaches in natural language processing: A survey. *Ai Open*, *3*, 71–90.

Li, Q., Peng, H., Li, J., Xia, C., Yang, R., Sun, L., Yu, P. S., & He, L. (2022). A survey on text classification: From traditional to deep learning. *ACM Transactions on Intelligent Systems and Technology (TIST)*, *13*(2), 1–41.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., & Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Liu, Z., Benge, C., & Jiang, S. (2023). Ticket-bert: Labeling incident management tickets with language models. *arXiv preprint arXiv:2307.00108*.

Mayer, C. W., Ludwig, S., & Brandt, S. (2023). Prompt text classifications with transformer models! an exemplary introduction to prompt-based learning with large language models. *Journal of Research on Technology in Education*, *55*(1), 125–141.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Min, J., McCoy, R. T., Das, D., Pitler, E., & Linzen, T. (2020). Syntactic data augmentation increases robustness to inference heuristics. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2339–2352.

OpenAI. (2024). Retrieved March 31, 2024, from https://openai.com/

Paramesh, S. (2023). Learning long-term dependencies for prediction of it incident category using lstm recurrent neural networks. *Journal of Data Acquisition and Processing*, *38*(2), 1156.

Paramesh, S., & Shreedhara, K. (2022). A deep learning based it service desk ticket classifier using cnn. *ICTACT Journal on Soft Computing*, *13*(1).

Pawar, C. S., & Makwana, A. (2022). Comparison of bert-base and gpt-3 for marathi text classification. *Futuristic Trends in Networks and Computing Technologies: Select Proceedings of Fourth International Conference on FTNCT 2021*, 563–574.

Pennington, J., Socher, R., & Manning, C. D. (2014). Glove: Global vectors for word representation. *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 1532–1543.

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., & Zettlemoyer, L. (2018). Deep contextualized word representations.

Qasim, R., Bangyal, W. H., Alqarni, M. A., & Almazroi, A. A. (2022). A fine-tuned bert-based transfer learning approach for text classification. *Journal of healthcare engineering*, *2022*.

Qi, Z. (2020). The text classification of theft crime based on tf-idf and xgboost model. *2020 IEEE International conference on artificial intelligence and computer applications (ICAICA)*, 1241–1246.

Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., et al. (2018). Improving language understanding by generative pre-training.

Regina, M., Meyer, M., & Goutal, S. (2020). Text data augmentation: Towards better detection of spear-phishing emails. *arXiv preprint arXiv:2007.02033*.

Revina, A., Buza, K., & Meister, V. G. (2020). It ticket classification: The simpler, the better. *IEEE Access*, *8*, 193380–193395.

Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019). Distilbert, a distilled version of bert: Smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.

Shachaf, G., Brutzkus, A., & Globerson, A. (2021). A theoretical analysis of fine-tuning with linear teachers. *Advances in Neural Information Processing Systems*, *34*, 15382–15394.

Shah, K., Patel, H., Sanghvi, D., & Shah, M. (2020). A comparative analysis of logistic regression, random forest and knn models for the text classification. *Augmented Human Research*, *5*(1), 12.

Sherstinsky, A. (2020). Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network. *Physica D: Nonlinear Phenomena*, *404*, 132306.

Sun, C., Qiu, X., Xu, Y., & Huang, X. (2019). How to fine-tune bert for text classification? *Chinese Computational Linguistics: 18th China National Conference, CCL 2019, Kunming, China, October 18–20, 2019, Proceedings 18*, 194–206.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser,, givenun=0. (2017). Attention is all you need. *Advances in neural information processing systems*, *30*.

Wei, J., & Zou, K. (2019). Eda: Easy data augmentation techniques for boosting performance on text classification tasks. *arXiv preprint arXiv:1901.11196*.

Xie, Q., Dai, Z., Hovy, E., Luong, T., & Le, Q. (2020). Unsupervised data augmentation for consistency training. *Advances in neural information processing systems*, *33*, 6256–6268.

Yu, B. (2008). An evaluation of text classification methods for literary study. *Literary and Linguistic Computing*, *23*(3), 327–343.

Zhao, B., Jin, W., Del Ser, J., & Yang, G. (2023). Chatagri: Exploring potentials of chatgpt on cross-linguistic agricultural text classification. *Neurocomputing*, *557*, 126708.

Zhao, W. X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., Min, Y., Zhang, B., Zhang, J., Dong, Z., et al. (2023). A survey of large language models. *arXiv preprint arXiv:2303.18223*.