

Darmstadt University of Applied Sciences

– Faculties of Mathematics and Natural Sciences
& Computer Science –

Satisfying Real-Time Requirements in Multi-Label Text Classification of Traveler Feedbacks with Transformer Models

Submitted in partial fulfilment of the requirements for
the degree of

Master of Science (M.Sc.)

by

Dennis Imhof

Matriculation number: 715862

First Examiner : Prof. Dr. Markus Döhring
Second Examiner : Prof. Dr. Antje Jahn
Issue date : 11.10.2022
Submission date : 26.03.2023

DECLARATION

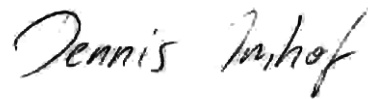
Ich versichere hiermit, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die im Literaturverzeichnis angegebenen Quellen benutzt habe.

Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht.

Die Zeichnungen oder Abbildungen in dieser Arbeit sind von mir selbst erstellt worden oder mit einem entsprechenden Quellennachweis versehen.

Diese Arbeit ist in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Darmstadt, 26. März 2023



Dennis Imhof

ABSTRACT

In this work we present a multi-step training and optimization scheme for real-time Multi-Label Classification (MLC) in the context of the long-distance personal rail transport industry. We are specifically dealing with anonymized traveler feedback texts submitted through multiple digital channels. To provide customers with context-dependent information, an instant response to their feedback has to be generated since anonymization prohibits the association of feedback and customer at a later point in time. Due to the sparse availability of multi-label annotated data, we utilize an ensemble of Large Language Model (LLM) binary classifiers trained on expert-annotated data to generate multi-label Pseudo Labels (PLs) for a large unlabeled dataset. A student LLM multi-label classifier is trained on the PL data and further latency-optimized through the application of neural network graph-optimization and quantization techniques.

We show, that we can distill the knowledge of an ensemble teacher model into a highly optimized student model with only a marginal loss of predictive power in a Multi-Label Classification problem incorporating eleven classes. We find only marginal performance degradation in the optimized student model, with the teacher model reaching a macro F1-score of 0.902 and the latency-optimized student model reaching a macro F1-score of 0.891 on a multi-label holdout testset. Contrastingly, the per-sample inference time of the student model can be reduced to 92ms on a commodity CPU, whereas the per-sample inference time of the ensemble model teacher depends on the slowest model in the ensemble and the degree of parallelization with around 500-700ms per-sample latency. Additionally, it can be shown that the multi-label student model even outperforms the binary ensemble teacher for some of the classes by utilizing learned label correlations the binary predictors have no access to. In a subsequent scalability experiment we extend the proposed workflow to a category selection of 82 categories and find that the student model reaches competitive performance on over 75% of the selected categories.

Keywords – Multi-Label Text Classification; Pseudo Labels; Latency; Transformer; ELECTRA; DistilBERT.

ZUSAMMENFASSUNG

In dieser Arbeit wird ein mehrstufiges Trainings- und Optimierungskonzept für Multilabel-Klassifikation in Echtzeit im Rahmen des Schienenpersonenfernverkehrs vorgestellt. Als Datengrundlage dienen anonymisierte Reisedenkenfeedbacks, die über verschiedene Digitalkanäle von KundInnen bereitgestellt werden. Die Klassifikation und die davon abgeleitete Antwort auf das Kundenfeedback muss in Echtzeit erfolgen, da durch die Anonymisierung der Feedbacks eine spätere Zuordnung und Antwort nicht mehr möglich ist. Da multi-label-annotierte Daten kaum verfügbar sind und sich der korrespondierende Annotationsprozess bei hohen Klassenanzahlen als enorm zeit- und ressourcenintensiv herausstellt, greifen wir auf ein Ensemble aus binären LLM Klassifikatoren zurück, die auf Basis von expertenannotierten Daten trainiert werden. Jenes Ensemble dient zur Erzeugung von PLs auf einem großen, ungelabelten Datenbestand. Anschließend wird ein multi-label *student model* mithilfe der PL-Daten trainiert und mithilfe von Netzwerkgraphoptimierung sowie Quantisierung latenzoptimiert.

Die Ergebnisse dieser Arbeit zeigen, dass ein Ensemble aus elf binären LLM Klassifikatoren ohne merklichen prädiktiven Performanzverlust in ein latenzoptimiertes *student model* distilliert werden kann. Hierbei weist das Ensemble ein makro F1-Evaluationsergebnis von 0.902 und das *student model* einen minimal geringeren Wert von 0.891 auf einem vorgehaltenen Testdatensatz auf. Gleichzeitig kann die per-Sample Inferenzzeit des *student models* auf gewöhnlicher Konsumentenhardware von den 500-700ms Inferenzzeit des Ensembles auf 92ms reduziert werden. Weiterhin zeigt sich, dass das multi-label *student model* auf einer Teilmenge der Labels eine bessere prädiktive Performanz erzielt als das Ensemblemodell, was auf die Ausnutzung von Labelkorrelationen zurückzuführen ist, die dem binären Ensemble nicht zur Verfügung stehen. In einem anschließenden Skalierbarkeitsexperiment wird die Kategorieauswahl auf insgesamt 82 Kategorien erweitert und gezeigt, dass durch das *student model* ohne weitere Optimierung der Trainingsdaten eine mit dem Ensemblemodell vergleichbare prädiktive Performanz auf über 75% der Klassen erreicht werden kann.

Stichworte – Multi-label Text Classification; Pseudo Labels; Latency; Transformer; ELECTRA; DistilBERT.

CONTENTS

| | | |
|----------|--|----------|
| I | Thesis | 1 |
| 1 | Introduction | 2 |
| 1.1 | Motivation | 2 |
| 1.2 | Methodology and Outline | 3 |
| 2 | Related work | 5 |
| 3 | Foundations: Multi-label classification with Attention-based Neural Networks | 10 |
| 3.1 | Multi-Label Classification | 10 |
| 3.1.1 | Probability Theory | 12 |
| 3.1.2 | Label dependence | 12 |
| 3.1.3 | Evaluation Measures for Classification | 16 |
| 3.2 | Multi-Label Data Improvement | 19 |
| 3.2.1 | Active Learning | 19 |
| 3.2.2 | Pseudo Labels | 21 |
| 3.2.3 | Label Imbalance | 23 |
| 3.3 | Attention-based Deep Neural Networks | 27 |
| 3.3.1 | Text representation | 27 |
| 3.3.2 | Transformer Encoder Architecture | 29 |
| 3.3.3 | Attention | 33 |
| 3.3.4 | Pre-Training and Fine-Tuning | 38 |
| 3.3.5 | Multi-Label Classification with Transformers | 40 |
| 4 | Foundations: Inference Time Optimization | 43 |
| 4.1 | Quantization | 43 |
| 4.1.1 | Symmetric vs. Asymmetric Quantization | 44 |
| 4.1.2 | Uniform vs. Non-Uniform Quantization | 45 |
| 4.1.3 | Post-Training Quantization vs. Quantization-Aware Training | 45 |
| 4.2 | Knowledge Distillation | 46 |
| 5 | Concept and experimental setup | 49 |
| 5.1 | Data | 50 |
| 5.1.1 | Binary-labeled training data | 50 |
| 5.1.2 | Multi-label data | 54 |
| 5.2 | Training, evaluation and prediction | 58 |
| 5.2.1 | Baseline Models | 60 |
| 5.2.2 | Large Language Models | 61 |
| 5.3 | Inference Latency Optimization | 61 |

| | | |
|-----------|--|-----------|
| 6 | Experimental results | 63 |
| 6.1 | Predictive performance | 63 |
| 6.1.1 | Binary classifiers by example of <i>Passenger Rights</i> | 64 |
| 6.1.2 | Multi-label classifiers | 68 |
| 6.1.3 | Scaling to higher numbers of categories | 73 |
| 6.2 | Inference latency optimization on CPU | 76 |
| 7 | Discussion | 81 |
| 7.1 | Conclusion | 83 |
| | Bibliography | 85 |
| II | Appendix | 95 |
| A | Figures | 96 |
| A.1 | Predictive | 97 |
| A.2 | Latency | 97 |

LIST OF FIGURES

| | | |
|-------------|--|----|
| Figure 3.1 | Three classification types with associated (positive) labels in green and non-associated (negative) labels in red. Each label box also shows a corresponding predicted probability assigned by a trained classifier. | 11 |
| Figure 3.2 | Unconditional label dependence (sub-)graph | 13 |
| Figure 3.3 | General <i>pool-based Active Learning</i> Workflow | 20 |
| Figure 3.4 | Effect of majority class undersampling on the optimal decision boundary with i.i.d. sample (top) and majority class undersampling (bottom) | 25 |
| Figure 3.5 | Transformer Encoder Architecture [Vas+17] | 30 |
| Figure 3.6 | Transformer Encoder Block [Ala18] | 32 |
| Figure 3.7 | Unrolled recurrent neural network (RNN) | 33 |
| Figure 3.8 | Self-Attention and Cross-Attention visualized with BertViz [Vig19] | 34 |
| Figure 3.9 | Scaled Dot-Product Attention (left) and Multi-Head Attention (right) [Vas+17] | 37 |
| Figure 3.10 | Different Self-Attention Heads visualized with BertViz [Vig19] | 38 |
| Figure 3.11 | Inputs used BERT | 39 |
| Figure 3.12 | <i>Binary Cross-Entropy Loss</i> and <i>Focal Loss</i> for different predicted probabilities \hat{p} and true label $y = 1$ | 42 |
| Figure 4.1 | Symmetric and Asymmetric Quantization | 45 |
| Figure 4.2 | Uniform and Non-Uniform Quantization | 45 |
| Figure 4.3 | Knowledge Distillation [Gou+21] | 47 |
| Figure 5.1 | End-to-End Pipeline | 49 |
| Figure 5.2 | Wordclouds for a subset of six categories | 52 |
| Figure 5.3 | Distribution of predicted probabilities on unlabeled data | 54 |
| Figure 5.4 | Fraction of samples with <i>Scaled Confidence Score</i> above threshold α | 55 |
| Figure 5.5 | Label correlation $P(Y)$ found in the pseudo-label multi-label data | 56 |
| Figure 5.6 | Early stopping induced by stagnating averaged F1-score improvement | 60 |
| Figure 6.1 | Evaluation curves over five cross validation folds on cross validation data $\mathcal{D}_{val_{cv}}$ and holdout validation data $\mathcal{D}_{val_{ood}}$ | 66 |
| Figure 6.2 | Influence of Positive Class Weight on model performance | 67 |

| | | |
|------------|---|----|
| Figure 6.3 | Multi-label XGBoost macro evaluation metrics on the holdout validation data $\mathcal{D}_{val_{ood}}$ corresponding to different hyperparameter configurations in a Parallel Coordinates Plot | 69 |
| Figure 6.4 | ELECTRA: F1 Score, Recall and Precision per class on test data | 70 |
| Figure 6.5 | ELECTRA: Precision-Recall Curves for test data | 70 |
| Figure 6.6 | F1-Scores and Positive Label Training Data Ratio in context of the binary classifiers, the multi-label classifier and respective difference | 74 |
| Figure 6.7 | Linear regression plot showing the true values y and fitted values \hat{y} corresponding to the difference in F1-Score between binary and multi-label models on the binary cross-validation data against the dependent variable x corresponding to the logarithmic ratio of the binary and multi-label label distributions. | 75 |
| Figure 6.8 | Per-Sample Latencies for XGBoost baseline models and different input sequence lengths | 77 |
| Figure 6.9 | Per-sample latency for different model architectures and sequence lengths | 78 |
| Figure 7.1 | Contextualized response based on the detection of category <i>Punctuality</i> | 84 |
| Figure A.1 | GLUE Top 10 Models | 96 |
| Figure A.2 | Per-Sample Inference Latency for different model type with input sequence length 20 | 97 |
| Figure A.3 | Inference latency with ONNX runtime | 98 |
| Figure A.4 | Inference latency for 8-bit quantized model and ONNX runtime | 98 |

LIST OF TABLES

| | | |
|-----------|--|----|
| Table 2.1 | Related Work Summary | 9 |
| Table 3.1 | Conditional and marginal probability distributions $\mathbf{p}_x(\mathbf{Y})$ and $\mathbf{p}_x(Y_i)$ [DCH10] | 15 |
| Table 3.2 | Confusion matrix for binary classification | 16 |
| Table 3.3 | Parameter count of Pre-Trained Transformer models | 39 |
| Table 5.1 | Token count statistics per category | 51 |
| Table 5.2 | Sample and label distribution within the binary training data of eleven categories | 53 |
| Table 5.3 | Size of intersections between binary-annotated datasets | 53 |
| Table 5.4 | Label distribution of the confidence-filtered pseudo-label multi-label data consisting of 37,621 samples | 55 |
| Table 5.5 | Execution environment specifications | 62 |
| Table 6.1 | Exemplary Regular Expressions filtering for feedbacks relevant to category <i>Passenger Rights</i> | 64 |
| Table 6.2 | Binary classifier improvement by dataset curation through batchwise <i>Active Learning</i> | 65 |
| Table 6.3 | Comparison of the predictive performance of different model architectures on the test data via F1-Score | 72 |
| Table 6.4 | OLS Regression Results | 75 |
| Table 6.5 | Macro F1-scores of binary and multi-label classifiers by choosing the best performing n categories for the multi-label model | 76 |
| Table 6.6 | Per-sample latency sequence length 250 | 79 |
| Table 6.7 | Per-sample latency sequence length 512 | 79 |
| Table A.1 | Comparison of the predictive performance of different model architectures on the holdout validation data via F1-Score | 97 |

LIST OF ACRONYMS

| | |
|--------|---|
| BCE | Binary Cross-Entropy |
| BERT | Bidirectional Encoder Representations from Transformers |
| BR | Binary Relevance |
| CBOW | Continuous Bag of Words |
| CNN | convolutional neural network |
| FNN | feedforward neural network |
| GAN | Generative Adversarial Network |
| GELU | Gaussian Error Linear Unit |
| GLUE | General Language Understanding Evaluation |
| i.i.d. | independent and identically distributed |
| LS | Labelset |
| LLM | Large Language Model |
| LSTM | Long Short-Term Memory |
| MLC | Multi-Label Classification |
| NLP | Natural Language Processing |
| NLU | Natural Language Understanding |
| ONNX | Open Neural Network Exchange |
| PL | Pseudo Label |
| RNN | recurrent neural network |
| SVM | support vector machine |
| SSL | Semi-Supervised Learning |
| TFIDF | Term Frequency Inverse Document Frequency |

Part I
Thesis

INTRODUCTION

Long distance personal rail transport encompasses a very broad set of services ranging from transport, booking and connection scheduling to information distribution as well as travel services like gastronomic offers, luggage logistics and multimedia services. All of which might be the subject of customer feedbacks submitted via multiple digital channels as anonymized free text. These textual feedbacks are a valuable resource to detect problems in service quality and present a low usage barrier for customers, which makes it possible to obtain them in relatively high volume. Additionally, the subject matter of such feedbacks typically refers to recent events, which makes it possible to react, mitigate or even solve specific problems in a timely fashion.

The investigation and proposition of such a detection infrastructure capable of accurately predicting customer intent is the focus of this work. Apart from a high predictive performance, a real-time latency is required to instantly respond to a customer's feedback since anonymization makes a delayed response impossible. Additionally, the prediction should be performed on CPU hardware to satisfy cost and scalability constraints.

1.1 MOTIVATION

The emergence of pre-trained large language models utilizing the transformer architecture, specifically the (self-)attention mechanism, has led to a massive shift in the field of natural language processing. The superior context awareness of transformer-based models in comparison to classical machine learning models and sequential deep learning models has enabled the handling of a wide range of difficult language tasks with high precision. In environments with high amounts of incoming unlabeled data and potentially high numbers of categories, manually annotating multi-label data quickly becomes infeasible. Textual customer feedbacks are often short and associated with only a small amount of labels. As such, constructing a viable multi-label training dataset requires labeling enormous amounts of i.i.d. sampled unlabeled data. However, if it was possible to train performant multi-label models based on intermediary pseudo-label data generated by individual binary classifiers, the annotation process could be optimized significantly. Each binary label generator model and corresponding training dataset could be optimized individually and multi-label training data could be generated in quantities only limited by the amount of unlabeled data available. We contemplate that the high learning capacity of [LLMs](#) makes these models well suited for the creation of the best possible pseudo-label data and at the same time makes them highly suitable as final multi-label

model learning from the provided pseudo-label data. However, the high learning capacity of these models comes at the price of relatively high resource requirements and a corresponding high inference latency on CPU hardware. Thus, in this thesis we investigate the predictive performance as well as the performance regarding the inference latency of LLMs trained on pseudo-label data. In more practical terms, we are interested in creating a workflow that

- can easily be adapted to different category selections,
- yields a model with high predictive performance and real-time latency,
- is cost- and energy-efficient.

The trending topics in customer feedbacks vary by season, political decisions, changes in services as well as local- and worldwide events. As such, we want a potential classification system to be highly adaptable. Another important constraint are the costs of such a classification system. Since dedicated GPU-hardware as well as GPU-capable cloud instances are significantly more expensive than CPU-hardware, we prioritize a deployment on the latter. Following, we present the research goals outlining the focus of this thesis.

Research goals

The following three research questions are the main driver behind this work:

- *Is it possible to train a performant multi-label model without expert-annotated multi-label data?*
- *Can a multi-label language model outperform multiple binary language models and thus, can the multi-label model implicitly use label correlations to its advantage?*
- *Can a high capacity multi-label language model reach an inference latency of sub 100ms on CPU hardware?*

1.2 METHODOLOGY AND OUTLINE

Following this introductory chapter, we present an overview of previous research related to this work, which touches the fields of *Multi-Label Classification*, *Neural Network Compression* as well as the research fields of *Active* and *Semi-Supervised Learning*. In the two succeeding theoretical chapters we lay out the necessary basis for this work. We begin by presenting the fundamentals of *Multi-Label Classification* in [Chapter 3](#) as well as challenges and approaches for the curation and handling of *Multi-Label Data* in [Section 3.2](#). In the second theoretical chapter, [Chapter 4](#), we provide an overview of techniques to reduce the memory footprint and specifically the inference latency

of LLMs. In [Chapter 5](#) we discuss the setup, implementation and execution of the experiments conducted in context of this work. We propose a multi-step training and optimization scheme for real time MLC in the context of the long-distance personal rail transport industry. Due to only sparsely available multi-label-annotated data, we utilize PL data generated by an ensemble of optimized LLM binary classifiers, which are themselves trained on expert-annotated data. The intermediary PL data is used to compress the ensemble of LLM binary classifiers into a single multi-label student model and serves as a form of *Knowledge Distillation*. The student model trained on PL data is then further latency-optimized through the application of neural network graph optimization and quantization techniques. Following in [Chapter 6](#), we evaluate the experimental results and conclude with a critical discussion regarding the success of our research in [Chapter 7](#).

RELATED WORK

In this chapter we provide an overview of existing works related to the research questions formulated in [Section 1.1](#).

Is it possible to train a performant multi-label model without expert-annotated multi-label data?

The incorporation of unlabeled data or synthesized data into the training process of machine learning models is an important part of self-training and semi-supervised learning. The first mentions of self-training date back to Scudder et al. [[Scu65](#)] where they propose first training a teacher model on labeled data and subsequently use the teacher model to generate pseudo-labeled data. The pseudo-label data in combination with the initial labeled data is used as training data for a student model. The combination of a small amount of labeled data and large amounts of unlabeled data is a common characteristic for semi-supervised learning methods. Bucila et al. investigate the compression of an ensemble of models into a single model by training the multi-label model on a mixture of expert-annotated data and pseudo data generated from the binary ensemble [[BCNM06](#)]. They use eight tabular datasets in their experiments and propose a synthetic data generation scheme to create pseudo-data. The mixture of real and synthetic data is used to compress an ensemble of base learners into a single model without significant loss of performance. However, they focus on binary classification with conventional feed forward neural networks learning from tabular data.

In context of our work, data generation is not necessary since large amounts of unlabeled data are available. There is high interest in solving language tasks with reduced amounts of labeled data. This is because the annotation process is expensive and resource-intensive. Mintz et al. present *distant* and *weak supervision* for text classification tasks to create pseudo-label data from unlabeled data by means of word-searches and regular expressions [[Min+09](#)]. The so created pseudo-labels, unfortunately, contain high amounts of noise, which negatively impacts model performance. In the worst case this leads to the model overfitting on incorrect labels and sub-par performance. To decrease the impact of label noise on the trained model, various mitigation approaches have been proposed. These can be divided into methods trying to improve the pseudo-label generation process as well as methods trying to filter the pseudo-label data after creation. Works focussing on an improvement of the label generation process incorporate meta information [[MZS20](#)], the learning order of samples [[MDS22](#)] or contextualized searches instead of word-based searches into the pseudo-label creation process [[MS20](#)]. Var-

ious approaches for the subsequent filtering of pseudo-labels with the goal of selecting the most informative data have been investigated. These encompass iterative approaches, clustering [Zhe+21] as well as confidence-based approaches [Riz+21], [Hua+21], [Lee+13] which are closely related to *Active Learning* methods [RB21], [NDH19]. A shared similarity of the referenced works is that they extend existing labeled data with unlabeled data by means of semi-supervised learning. In our approach, however, we explicitly avoid using any expert-labeled multi-label training data. The initial training data for the teacher ensemble is a set of disjunct, binary-labeled datasets and the training data for the final compressed multi-label model is completely based on uncertainty-filtered pseudo-label data.

Key Results

Training models on a small set of labeled data and high amounts of unlabeled data is a common approach in *semi-supervised learning*. Given the constraints of disjunct binary training data, we completely cut out the labeled data in the training of the student.

Does a multi-label language model outperform multiple binary language models and thus, can the multi-label model implicitly use label correlations to its advantage?

There are several factors on which the aforementioned research question depends. For one, the predictive performance of the multi-label transformer model is directly tied to the quality of the pseudo-label data used as training data. Bai et al. find that early stopping is important to stop the model in training from overfitting on noisy labels [SGM22]. Zhu et al. state that early stopping should be induced if the models performance degrades on a clean validation set [Zhu+22]. Another important factor is the modeling of the multi-label problem regarding correlations and dependencies between labels or set of labels. Dmbczynski et al. show that explicitly modeling label dependencies by utilizing a higher order multi-label modeling approach leads to a drastically different optimization problem than the utilization of the binary relevance first order approach [DCH10]. They find that a binary relevance approach might be preferable if the performance of individual classes is more important than an exact match of all labels. Label dependencies can explicitly be modelled with the label powerset approach [TV07], which, however, leads to high label sparsity and an exponential increase in required data. *Classifier chains* also fall into the category of higher order methods. They iteratively incorporate label subsets into the feature space and have successfully been applied to multi-label classification on short textual data [DWH12], [Sch+14]. In context of large deep learning models, however, Nam et al. find that classifier chains come with prohibitively high resource overhead [Nam+19]. Different approaches for modeling label semantics and dependencies explicitly with transformer models have been proposed that incorporate meta-data and hierarchical information [Zha+21], label subset

tagging [Liu+17] or add an additional correlation output module to capture possible label correlations [Xun+20]. Devlin et al. [Dev+18] as well as Brown et al. [Bro+20] show that the unsupervised pre-training stages applied to BERT, GPT and adjacent transformer architectures lead to high generalization capabilities, intrinsic context awareness and state-of-the-art results on various benchmark experiments. As such, pre-trained LLMs should be able to implicitly utilize label correlations.

Key Results

Pre-trained *transformer* models are highly context-aware. We hypothesize, that they should be capable to utilize label correlations in the input data without explicit modelling of label dependencies.

Can a high capacity multi-label language model reach an inference latency of sub 100ms on CPU hardware?

The size of a LLM, or rather the number of utilized parameters, directly influences the potential predictive performance as well as inference latency of the model. Brown et al. show that efficient context utilization and task performance directly correspond to model size [Bro+20]. In a real-time context where predictive performance and low inference latency are of equal importance, model optimizations have to be performed that at best reduce inference latency without degrading the predictive performance of the model. Han et al. investigate the efficiency of compressed neural networks regarding inference speed and energy consumption on embedded devices with resource constrained hardware. [Han+16] They show that the memory access required to fetch the parameters of a neural network causes energy costs two magnitudes larger than arithmetic operations executed with said parameters and consequently also significantly contributes to the total inference latency. Methods with the goal of increasing the execution performance of neural networks typically try to optimize the utilization of hardware in a specific execution environment, optimize the memory layout of the model or reduce the number of floating point operations as well as the number of memory accesses required for a forward-pass.

Different areas of research focus on the creation of efficient neural network architectures and the modification of existing architectures for increased efficiency. One of these areas is known as *knowledge distillation* [HVD15], where a smaller, more memory-efficient student model learns from a larger teacher model. Another field known as *pruning* focusses on the reduction of employed weights in neural network models. This reduction is typically performed during or post-training [GDA20], [WWL20]. It is assumed that a neural network contains high degrees of redundancy and that there exists an optimal sub-network equally performant as the fully specified parent network. This is also known as the *Lottery Ticket Hypothesis* [Mor+19],

[BH20] and serves as theoretical underpinning for as to why pruning and the sparcification of neural networks works. Sanh et al. present *DistilBERT*, a pruned and knowledge distilled successor of *BERT*, with a size reduction of 40% and a 60% inference speedup compared to *BERT*. The execution of sparse neural networks is another active area of research connecting pruning and hardware-efficient neural network layouts [Lag+21], [Hoe+21]. Different frameworks and compiler techniques have emerged and evolved with the focus of optimizing neural network graphs on specific execution environments [Rot+18], [Lat+21]. Lastly, the research area known as *quantization* focusses on a more memory-efficient representation of neural networks by utilizing 16bit floating-point values (also known as half-precision) or integer values to represent the weights of the neural network [Gup+15], [Zaf+19], [Jac+18]. Kim et al. find that an integer-transformed *BERT* shows a speedup of up to 3 times of that of the original 32bit floating-point formulation [Kim+21].

The average per-sample latency highly depends on the utilized hardware and execution environment. As such, publications typically report relative latency improvements instead of hardware-specific absolute values. *Hugging-Face* report that by employing multi-threading on a high performance CPU, a sub-100ms per-sample latency with *BERT* in a PyTorch runtime is achievable ¹.

Key Results

Achieving a sub-100ms per-sample latency should be achievable with high performance CPU hardware. We investigate if the previously presented optimization techniques enable us to perform sub-100ms inference on commodity hardware without significant predictive performance loss.

¹ <https://huggingface.co/blog/bert-cpu-scaling-part-1> (last visited on 26.03.2023)

Summarization and overview

| Topic | Publication | Keywords |
|-------------------|-------------|--|
| Pseudo-Labels | [BCNM06] | Ensemble compression, feed-forward neural networks, 8 binary tabular datasets, only pseudo-labels |
| | [Lee+13] | Multi-Class, MNIST dataset, Entropy Regularization |
| | [Riz+21] | Uncertainty selection, model calibration, CIFAR-10, CIFAR-100, Pascal VOC and UCF-101 datasets |
| Active Learning | [LG94] | Uncertainty sampling, text classification, AP newswire data subset |
| | [RB21] | Noisy labels, uncertainty sampling synthetic data as well as letter-binary and covertype datasets |
| Transformer | [Vas+17] | Transformer architecture, attention mechanism, WMT 2014 dataset |
| | [Dev+18] | <i>BERT</i> , pre-training for transformers, GLUE datasets |
| | [Cla+20] | <i>ELECTRA</i> , improved pre-training scheme, GLUE datasets |
| Model compression | [HVD15] | Knowledge distillation, MNIST & JFT image datasets |
| | [San+19] | Distilbert, knowledge distillation for transformers, GLUE dataset |
| | [Kim+21] | 8-bit quantized <i>BERT</i> , GLUE datasets |

Table 2.1: Related Work Summary

FOUNDATIONS: MULTI-LABEL CLASSIFICATION WITH ATTENTION-BASED NEURAL NETWORKS

This chapter focuses on the theoretical foundations of [MLC](#) and the application of attention-based deep neural networks to [MLC](#) problems. In [Section 3.1](#) we develop the necessary background to understand multi-label classification in a probabilistic context. In the subsequent [Section 3.2](#), we examine challenges and respective solutions specific to multi-label data imbalance and sparsity. The architecture of attention-based deep neural networks, whose predictive performance in multi-label classification is another core subject of this work, is the focus of [Section 3.3](#).

Notations

Throughout this work we will use the following notations.

- \mathbf{X} - Multivariate random variables
- X - Univariate random variables
- \mathbf{x} - Realizations of multivariate a random variable
- x - Realizations of a univariate random variable
- \mathcal{X} - Mathematical space

3.1 MULTI-LABEL CLASSIFICATION

Classification is the task of associating an observation

$$\mathbf{x} = (x_1, \dots, x_p) \in \mathcal{X} \quad (3.1)$$

with one or multiple discrete labels λ_i with

$$L = \{\lambda_1, \dots, \lambda_m\} \subseteq \mathcal{L} \quad (3.2)$$

being a label subset from a finite set of labels \mathcal{L} . We call $\mathcal{X} \subseteq \mathbb{R}^p$ the p -dimensional input or feature space. Exemplary input data \mathbf{x} might be text, sound, image or tabular data and the associated labels λ_i could represent music genres, emotions or even chemical elements. We differentiate between three different kinds of classification tasks, which are

- *binary* classification,

- *multi-class* classification
- and *multi-label* classification.

A visualization of the three different types of classification problems is shown in [Figure 3.1](#).

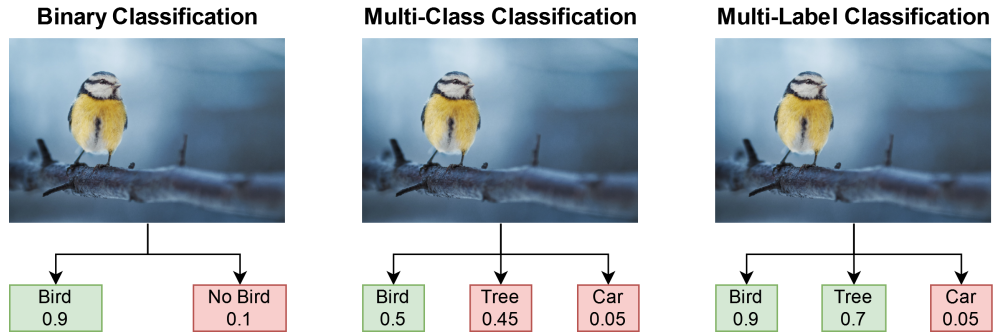


Figure 3.1: Three classification types with associated (positive) labels in green and non-associated (negative) labels in red. Each label box also shows a corresponding predicted probability assigned by a trained classifier.

In *binary* and *multi-class* classification an observation \mathbf{x} is associated with exactly one label $\lambda_i \in \mathcal{L}$. This can be seen in the left and center panel of [Figure 3.1](#) where exactly one of the possible labels is positive. *Binary* classification can be viewed as special case of *multi-class* classification with the label set \mathcal{L} consisting of exactly two labels ie. $|\mathcal{L}| = 2$. *Multi-label* classification on the other hand permits the association of an observation \mathbf{x} with an arbitrary subset of labels $L \subseteq \mathcal{L}$ and *multi-class* classification in turn can be viewed as a special case of *multi-label* classification with $|L| = 1$. Many classification problems are multi-label in nature. A movie or a piece of music might be associated with multiple genres, an image might contain multiple objects we want to classify and a text document might span various relevant topics.

We can approach a classification problem by utilizing methods of supervised learning where a statistical model is fitted to training data

$$\mathcal{D}_{train} = \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})_{i=1}^n\} \quad (3.3)$$

with the goal of estimating an often complex underlying functional mapping

$$f : \mathbf{x} \rightarrow \mathbf{y}. \quad (3.4)$$

The trained statistical model and thereby the learned or estimated mapping \hat{f} can subsequently be used to predict the label associations of unseen data

$$\hat{\mathbf{y}} = \hat{f}(\mathbf{x}). \quad (3.5)$$

We use binary vectors $\mathbf{y} \in \mathcal{Y}$ and $\hat{\mathbf{y}} \in \mathcal{Y}$ to represent the sets of true label associations L and predicted label associations \hat{L} of an observation \mathbf{x} . We call \mathcal{Y} the output or label space. Both binary vectors are of dimension $|\mathcal{L}|$ with

$$\mathbf{y}, \hat{\mathbf{y}} \in \mathcal{Y} \quad \mathcal{Y} = \{0, 1\}^{|\mathcal{L}|} \quad (3.6)$$

where the entry y_i at index i indicates the association or non-association of the observation \mathbf{x} with label λ_i ie.

$$\begin{aligned} y_i = 0 &\Leftrightarrow \lambda_i \notin L \\ y_i = 1 &\Leftrightarrow \lambda_i \in L. \end{aligned} \quad (3.7)$$

3.1.1 Probability Theory

The training data $\mathcal{D}_{train} = \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})_{i=1}^n\}$ is assumed to be an independent and identically distributed (i.i.d.) sample from an unknown joint probability distribution $\mathbf{p}(\mathbf{X}, \mathbf{Y})$ on $\mathcal{X} \times \mathcal{Y}$ [Dem+12] with $\mathbf{X} = (X_1, \dots, X_p)$ and $\mathbf{Y} = (Y_1, \dots, Y_{|\mathcal{L}|})$ being multivariate random variables. If we had access to the joint probability distribution, an unseen observation \mathbf{x} could be classified by finding the mode of the conditional probability distribution $\mathbf{p}_{\mathbf{X}}(\mathbf{Y})$:

$$\begin{aligned} \mathbf{c}_{opt} &= \arg \max_{\mathbf{c}} \mathbf{p}_{\mathbf{X}}(\mathbf{Y}) \\ &= \arg \max_{\mathbf{c}} \mathbf{p}(\mathbf{Y} = \mathbf{c} | \mathbf{X} = \mathbf{x}) \\ &= \arg \max_{\mathbf{c}} \frac{\mathbf{p}(\mathbf{X} = \mathbf{x}, \mathbf{Y} = \mathbf{c})}{\mathbf{p}(\mathbf{X} = \mathbf{x})} \end{aligned} \quad (3.8)$$

Here $\mathbf{p}(\mathbf{X} = \mathbf{x})$ is the marginal distribution of \mathbf{X}

$$\mathbf{p}(\mathbf{X} = \mathbf{x}) = \sum_{\mathbf{y} \in \mathcal{Y}} \mathbf{p}(\mathbf{X} = \mathbf{x}, \mathbf{Y} = \mathbf{y}) \quad (3.9)$$

which can be calculated by summing the joint probability distribution over the support of the discrete multivariate random variable \mathbf{Y} . Modeling the joint probability distribution directly falls into the area of generative modeling, which is a very difficult task given high dimensional data ([Biso6], p.44f). Though, if we're mainly interested in discriminative results, modeling the full joint probability distribution is not necessary. Instead we try to model the joint conditional probability distribution $\mathbf{p}(\mathbf{Y} | \mathbf{X} = \mathbf{x})$ or the individual marginal probability distributions $\mathbf{p}(Y_i | \mathbf{X} = \mathbf{x})$ depending on our specific goals and assumptions regarding label dependence.

3.1.2 Label dependence

In MLC we differentiate between two forms of statistical dependence between labels. These are *unconditional* and *conditional* label dependence [Dem+10]. Unconditional label dependence is a dependence between labels Y_i regardless of any association with a specific instance \mathbf{x} . As an example, if the task was to assign labels to products and the label set contained the labels *rare*,

novel, *high production cost* and *expensive*, the presence of any of the labels *rare*, *high production cost* and *novel* would make the occurrence of the label *expensive* highly likely regardless of an association with a specific product. Though, we assume a dependence of the label *expensive* on the former labels as shown in Figure 3.2.

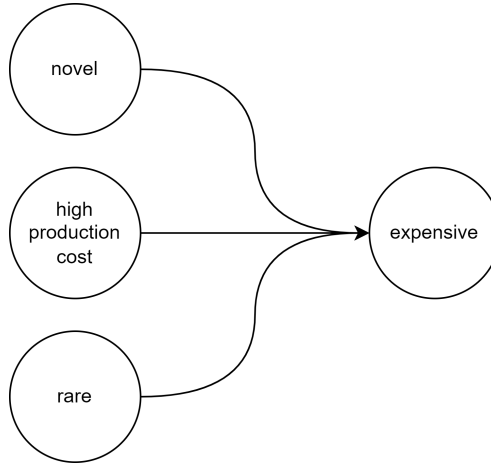


Figure 3.2: Unconditional label dependence (sub-)graph

In the most extreme case unconditional dependence between labels could mean that the set of labels contains highly redundant labels. Formally, unconditional label *independence* is defined as

$$\mathbf{p}(\mathbf{Y}) = \prod_{i=1}^m \mathbf{p}(Y_i) \quad (3.10)$$

and thus unconditional label dependence exists, if the equality between the joint label probability distribution and the product of the marginal label probability distributions does not hold. As mentioned in Chapter 2, there are various works concerned with the detection and incorporation of unconditional label dependence into the modeling process.

Another form of label dependence is called *conditional label dependence*. Conditional label *independence* is formally defined as the equivalence between the joint conditional probability distribution of \mathbf{Y} and the product of its conditional marginal probability distributions

$$\mathbf{p}_{\mathbf{x}}(\mathbf{Y}) = \prod_{i=1}^m \mathbf{p}_{\mathbf{x}}(Y_i). \quad (3.11)$$

Modeling under the strong assumption that conditional label independence holds, leads to the Binary Relevance (BR) approach to multi-label classification, a first order approach where the classification of each label is viewed as individual binary classification task. Conversely, the incorporation of conditional label dependence into the modeling task leads to second and higher order approaches to MLC.

Modeling Strategies for Label Dependence

FIRST ORDER STRATEGIES: Each label is treated in an individual binary classification problem. Label dependencies are not explicitly integrated into the modeling process. This approach enables the use of ensembles of individually trained models as well as the use of multi-output models optimized with a loss function that doesn't take any form of label dependence into account.

SECOND ORDER STRATEGIES: This approach to MLC incorporates pairwise label dependence into the modeling process. This can be achieved by the usage of pairwise ranking loss-functions. Pairwise approaches come with a quadratic increase of computational complexity since each pair of labels has to be taken into account.

HIGHER ORDER STRATEGIES: These strategies try to explicitly include all dependencies between labels into the modelling process. A common approach is the Labelset (LS) problem transformation approach. Here each label subset is viewed as an individual class and the multi-label problem is transformed into a multi-class problem. Worst case, this means that the computational complexity increases exponentially from L to 2^L . There are approaches like the *Pruned Set* approach [RPHo8] that try to alleviate the high computational complexity by pruning infrequent sets of labels.

Dembczyński et al. compared the *first order BR* approach with the *higher order LS* approach and found that the performance of each highly depends on the choice of loss function [Dem+10] and ultimately on the specific goal we try to optimize for. They showed that the *BR* approach, which tries to predict the marginal conditional modes

$$\arg \max_i \mathbf{p}_x(Y_i),$$

on average performed better with respect to the *Hamming Loss*

$$\mathcal{L}_{\text{Hamming}}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{m} \sum_{i=1}^{|\mathcal{L}|} \mathbb{I}[y_i \neq \hat{y}_i], \quad (3.12)$$

which is defined as the fraction of incorrectly predicted labels. The *LS* approach on the other hand performed better with respect to the *Subset 0/1-Loss*, which is defined as

$$\mathcal{L}_{0/1}(\mathbf{y}, \hat{\mathbf{y}}) = \mathbb{I}[\mathbf{y} \neq \hat{\mathbf{y}}] \quad (3.13)$$

and whose minimization corresponds to the minimization of the loss w.r.t the joint conditional probability distribution $\mathbf{p}_x(\mathbf{Y})$ [DWH12]. The *Subset 0/1-*

Loss is much more rigid than the *Hamming Loss* and equally penalizes any divergence of the predicted set of labels from the true set of labels.

Which strategy to choose?

As briefly discussed in [Section 3.1.1](#), each of the different **MLC** problem modeling strategies comes with its own set of advantages and disadvantages. The most notable factors thereby tend to be:

1. Strength of assumptions and degree of problem simplification.
2. Data and compute resource requirements.

We first look at item (1) by comparing the predicted set of labels given a **BR** approach utilizing the marginal conditional modes and a **LS** approach utilizing the joint conditional mode. In [Table 3.1](#) we show an exemplary joint conditional probability distribution $\mathbf{p}_x(\mathbf{Y})$ as well as the marginal conditional probability distributions $\mathbf{p}_x(Y_i)$ for two random variables Y_1 and Y_2 , which represent the set of labels.

| $\mathbf{p}_x(\mathbf{Y})$ | 0 | 1 | $\mathbf{p}_x(Y_2)$ |
|----------------------------|-----|-----|---------------------|
| 0 | 0.0 | 0.4 | 0.4 |
| 1 | 0.3 | 0.3 | 0.6 |
| $\mathbf{p}_x(Y_1)$ | 0.3 | 0.7 | 1 |

Table 3.1: Conditional and marginal probability distributions $\mathbf{p}_x(\mathbf{Y})$ and $\mathbf{p}_x(Y_i)$ [[DCH10](#)]

We can see the problematic conclusion caused by the false assumption of conditional label independence. The mode of the joint conditional probability distribution is found at

$$\mathbf{p}_x(Y_1 = 0, Y_2 = 1) = 0.4,$$

whereas the product of the marginal modes is calculated as

$$\mathbf{p}_x(Y_1 = 1)\mathbf{p}_x(Y_2 = 1) = 0.7 \cdot 0.6 = 0.42$$

and thus, the **BR** approach would lead to the false conclusion that both labels should be associated with observation \mathbf{x} . Under the strict constraints of the *Subset 0/1-Loss* this would mean that the classification is wrong. However, in context of the *Hamming Loss* the classification is 50% correct and 50% wrong. Now, if we imagine the classification of an observation with one misclassified label $\hat{y}_i \neq y_i$ out of $|\mathcal{L}| = 100$ labels, the *Subset 0/1-Loss* would still indicate a wrong classification. Contrastingly, the *Hamming Loss* would penalize only the misclassified label surmounting in a total loss of 0.01, which might be a more desirable outcome.

As mentioned in [Section 3.1.1](#), *second* and *higher order strategies* show quadratic and exponential computational complexity in L . Even worse, this quadratic and exponential increase not only applies to the computational complexity, but also to the amount of required, labeled training data. To put things into perspective, an exemplary multi-label classification problem concerned with the prediction of a set of labels with $|\mathcal{L}| = 10$ would require the counts of observations shown in [Equation 3.14](#) to at least provide one positive example of a label, n_F , label pair, n_S , and label subset, n_H , corresponding to the *first*, *second* and *higher order strategy*, respectively.

$$\begin{aligned} n_F &= |\mathcal{L}| & n_S &= \binom{|\mathcal{L}|}{2} & n_H &= 2^{|\mathcal{L}|} \\ &= 10 & &= 45 & &= 1024 \end{aligned} \quad (3.14)$$

Additionally, the correctness of a specific label subset might not matter as much as the average partial correctness of labels for the specific use case at hand. Thus, the [BR](#) approach, optimizing the *Hamming Loss* or a surrogate loss like the *Binary Cross Entropy Loss*, which will be discussed in [Section 3.1.3](#), is the the prioritized strategy in context of this work.

3.1.3 Evaluation Measures for Classification

Many evaluation measures for single- and multi-label classification can be derived from a *contingency table* also known as *confusion matrix*. An exemplary contingency table is shown in [Table 3.2](#). *True Positives (TP)*, *False Positives (FP)*, *False Negatives (FN)* and *True Negatives (TN)* are positive integers (counts) where *true* and *false* signal the correctness of the prediction whereas *positive* and *negative* signal the value of the prediction.

| | | True value | | Total |
|-----------------|----------|------------|-----------|-----------|
| | | Positive | Negative | |
| Predicted value | Positive | TP | FP | $TP + FP$ |
| | Negative | FN | TN | $FN + TN$ |
| Total | | $TP + FN$ | $FP + TN$ | N |

Table 3.2: Confusion matrix for binary classification

Common evaluation measures for binary classification that can be derived from the counts found in a contingency table are *Accuracy*, *Precision*, *Recall* and the *F1-Score* among others.

$$\begin{aligned}
Accuracy &= \frac{TP + TN}{N} \\
Precision &= \frac{TP}{TP + FP} \\
Recall &= \frac{TP}{TP + FN} \\
F1-Score &= \frac{2}{Precision^{-1} + Recall^{-1}}
\end{aligned} \tag{3.15}$$

Tangible explanations of these measures are as follows:

ACCURACY also known as *Hit-Rate*, measures the fraction of correct predictions. This is a good general measure if the distribution of labels is balanced and both, positive as well as negative predictions, are equally important.

PRECISION represents the quality of positive predictions. This is an important measure for recommender systems. Here being sure about the relevance of an advertised product is often more important than exhaustively advertising all possibly relevant products to a customer.

RECALL or *sensitivity* is measuring the fraction of true positive observations the classifier has successfully detected. This is a very important measure in medical classification problems where we want to be sure to detect a specific disease.

F1-SCORE is the harmonic mean of the *Precision* and *Recall* measures and thus takes into account the quality of positive predictions as well as the ability to detect positives. This will be the reference measure in context of this work, since we are confronted with major label imbalance and prioritize positives over negatives.

To apply these measures to a **MLC** problem, we can use *micro* and *macro* averages. A *micro* average is calculated on the summed confusion matrix or contingency table over all classes, which makes it favor majority classes. A *macro* average on the other hand is calculated as unweighted average of the individual class metrics and thereby favors minority classes. Favor in this context means assigning a disproportionately high weight to the respective class. For a general binary measure $B(TP, FP, FN, TN)$ the calculation of the respective *micro* and *macro* measures is as follows

$$\begin{aligned}
B_{micro} &= B\left(\sum_{i=1}^{|\mathcal{L}|} TP_i, \sum_{i=1}^{|\mathcal{L}|} FP_i, \sum_{i=1}^{|\mathcal{L}|} FN_i, \sum_{i=1}^{|\mathcal{L}|} TN_i\right) \\
B_{macro} &= \frac{1}{|\mathcal{L}|} \sum_{i=1}^{|\mathcal{L}|} B(TP_i, FP_i, FN_i, TN_i)
\end{aligned} \tag{3.16}$$

The exemplary formulation of the *micro* and *macro* *F1-Scores* is shown in Equation 3.17.

$$\begin{aligned}
 F1_{micro} &= \frac{2 \cdot \sum_{i=1}^{|\mathcal{L}|} TP_i}{\sum_{i=1}^{|\mathcal{L}|} (2 \cdot TP_i + FP_i + FN_i)} \\
 F1_{macro} &= \frac{1}{|\mathcal{L}|} \sum_{i=1}^{|\mathcal{L}|} \frac{2 \cdot TP_i}{2 \cdot TP_i + FP_i + FN_i}
 \end{aligned}
 \tag{3.17}$$

As mentioned before, we would like to optimize our multi-label classifier with respect to the *F1-measure*. Unfortunately, the binary measures derived from a contingency table are not suitable in the role of a loss function. This is because such a loss function would not be continuous and differentiable. Thus, it would be ill-suited in conjunction with gradient descent methods used for the training of neural networks. As a result, surrogate loss functions that are presented in Section 3.3 will be used for the training. The binary evaluation measures shown in this section are used for continuous evaluation and as early stopping criterion for the training progress.

3.2 MULTI-LABEL DATA IMPROVEMENT

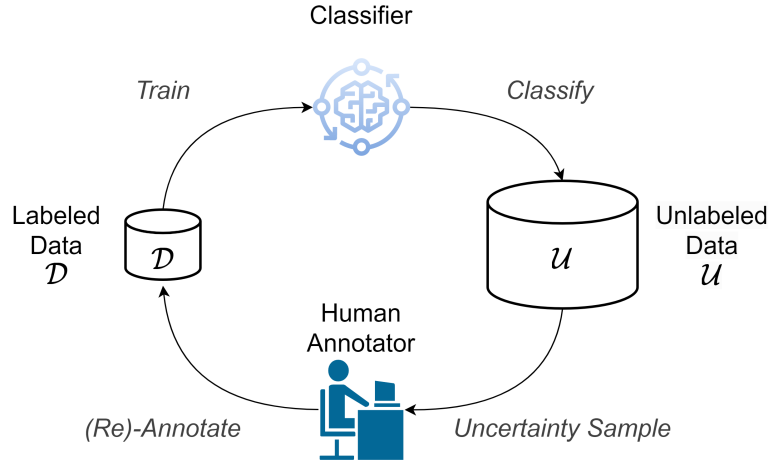
The curation of adequate datasets for the training and validation of multi-label classifiers poses various difficulties amplified by the size of the set of relevant labels. As shown in [Section 3.1.1](#), if we want to ensure an exhaustive selection of labels in the multi-label training data, for *higher order strategies* like the [LS](#) approach, we are required to provide a number of observations exponential in \mathcal{L} , which quickly becomes infeasible in practice. Additionally and given the concrete problem of customer feedback classification at hand, historic data used as training and validation data becomes stale over time, which means that the historic data isn't a good representation for the more recent data anymore. This phenomenon is known as *Data Drift* and leads to a degradation of model performance. For a real time classification system exclusively dealing with the most recent data, detecting and combating *Data Drift* and thus constantly updating the underlying data is necessary.

This section touches upon different challenges in creating, curating, updating and classifying multi-label data and discusses strategies for managing said challenges. In [Section 3.2.1](#) we will first look at the optimization of binary labeled data and corresponding binary classification models by utilizing *Active Learning* methods. Following in [Section 3.2.2](#), we discuss the creation of *Pseudo-Label* multi-label data used in a *Knowledge Distillation* scheme, where multi-label classification models are trained by learning from an ensemble teacher model consisting of optimized binary classifiers. We conclude this section by looking at ways to handle data imbalance in multi-label data in [Section 3.2.3](#).

3.2.1 Active Learning

Active learning is a field of machine learning concerned with optimizing the experimental design. A key idea of *Active Learning* is that the amount of data and computational resources needed to train a machine learning model can be minimized by efficiently curating the training data. Therefore the model is integrated into the data selection process. Through the usage of some predefined measure of uncertainty the model selects or *queries* observations from a set of unlabeled data that it is the least sure about. These observations are then labeled by a human expert and added to the labeled training data. An extensive summary of research and application areas concerning *Active Learning* was presented by Prince et al. [[Prio4](#)].

We are especially interested in *pool-based active learning* for sequential data. Here a large pool of unlabeled data is iteratively queried for the most informative observations. This sampling procedure is called *Uncertainty Sampling* [[LG94](#)] and the general workflow is shown in [Figure 3.3](#).

Figure 3.3: General *pool-based Active Learning* Workflow

To quantify the uncertainty of a probabilistic model with parameters θ given a specific observation \mathbf{x} , different measures in form of a score function $s(\theta, \mathbf{x})$ can be applied [NSH22]. Typical measures for binary and multi-class classification are the Shannon entropy [Sha48]

$$s(\theta, \mathbf{x}) = - \sum_{y \in \mathcal{Y}} p_{\theta}(y|\mathbf{x}) \log p_{\theta}(y|\mathbf{x}), \quad (3.18)$$

the least confidence measure

$$s(\theta, \mathbf{x}) = 1 - \max_{y \in \mathcal{Y}} p_{\theta}(y|\mathbf{x}). \quad (3.19)$$

and the *smallest margin* measure

$$s(\theta, \mathbf{x}) = p_{\theta}(y_m|\mathbf{x}) - p_{\theta}(y_n|\mathbf{x}) \quad (3.20)$$

with

$$\begin{aligned} y_m &= \arg \max_{y \in \mathcal{Y}} p_{\theta}(y|\mathbf{x}) \\ y_n &= \arg \max_{y \in \mathcal{Y} \setminus y_m} p_{\theta}(y|\mathbf{x}). \end{aligned}$$

In case of the *entropy* and *least confidence* measures, we are looking for the examples \mathbf{x} with the highest score indicating the highest degree of *uncertainty*. In case of the *smallest margin* measure, we are selecting the observations with the lowest score, indicating that the probabilistic model has the most difficulty deciding between label y_m and y_n .

Scaled confidence score

In this work we will employ a scaled version of the least confidence measure. This modified version takes decision thresholds deviating from the often employed decision threshold $t = 0.5$ into account. The decision threshold is

the value used to decide between a negative and positive prediction given the probability output of a binary classifier. As such, the hard thresholding or binarization of a classifiers output can be formalized as

$$f_{bin}(\hat{p}, t) = \mathbb{I}[\hat{p} \geq t] \quad (3.21)$$

We employ decision threshold tuning to optimize the *F1-Score* as described in [Section 3.2.3](#). To account for the imbalance between the probability regions below and above the decision threshold $t \neq 0.5$, we scale the least confidence score leading to the following definition

$$s(\hat{p}, t) = 1 - \frac{\hat{p}}{2} \left(\frac{\mathbb{I}[\hat{p} < t]}{t} + \frac{\mathbb{I}[\hat{p} \geq t]}{1-t} \right). \quad (3.22)$$

The *scaled least confidence score* as well as the *macro* formulation thereof will also play an important role in the selection of *Pseudo Label* data, which we will explain in [Section 3.2.2](#).

3.2.2 Pseudo Labels

The manual annotation of large amounts of data is time-consuming, often costly and error-prone depending on the domain knowledge of the annotator(s) as well as depending on the subjectivity of the classification problem at hand. It would be highly desirable, if we could reduce the manual annotation efforts to a necessary minimum. Sparsity of labeled data as well as the aforementioned challenges posed by manual annotation efforts are part of the motivation for research fields known as *Self-Supervised Learning*, *Weakly Supervised Learning* and *Semi-Supervised Learning* that try to incorporate unlabeled data into the training process.

Different subfields of *Self-Supervised Learning* have emerged like *Consistency Regularization*, where samples are augmented with minor *perturbations* and an inconsistent model output given the original and the augmented input is penalized by an auxiliary loss function [BAP14][SJT16]. Another approach is known as *Co-Training*, where multiple classifiers are trained with the goal of having each classifier learn unique representations of the data. The consistency of their predictions is evaluated and each classifier can be used to generate weakly labeled data for the remaining classifiers [BM98] [Qia+18]. An extensive overview of Semi-Supervised Learning (SSL) techniques and challenges can be found in [TGH15] and [VEH20]. Generally, methods of *Semi-Supervised Learning* make the following assumptions about the data:

CLUSTER ASSUMPTION: The input data is assumed to cluster in high density regions corresponding to the classes of the classification problem. Consequently, optimal decision boundaries should lie in low-density regions

SMOOTHNESS ASSUMPTION: If two points $x_1, x_2 \in \mathcal{X}$ are close in the *feature space* \mathcal{X} , they should also be close in the *output space* \mathcal{Y} , which

means that they typically should be assigned the same class. *Consistency Regularization* builds upon this property.

MANIFOLD ASSUMPTION: We assume that the input data residing in the high-dimensional *feature space* $\mathcal{X} = \mathbb{R}^p$ is concentrated on a lower-dimensional space or *manifold*. Under this assumption, it should be possible to separate dense regions or clusters with lower dimensional functions.

In this work we focus on a method of SSL called *Pseudo Labeling* or *Weak Labeling*. *Pseudo Labels* are labels that are assigned to unlabeled data by an automatic mechanism, which might be a rule based system [Min+09] or a trained classifier [Riz+21][MA20], instead of being assigned by a human expert. Additionally, an *uncertainty measure* based on the estimated probabilities is used to estimate the quality of the generated labels.

Given an observation \mathbf{x}_i , a decision threshold t and a binary probabilistic classifier p_θ , a hard *pseudo-label* \tilde{y}_i can be generated by applying the following rule:

$$\tilde{y}_i = \mathbb{I}[p_\theta(\mathbf{x}) \geq t] \quad (3.23)$$

To quantify the classifiers uncertainty with respect to \tilde{y}_i , we can utilize the (*scaled*) *least confidence measure* $s(\theta, \mathbf{x})$ presented in Section 3.2.1. Given the selection thresholds τ_p and τ_n for positive and negative examples, respectively, we can select the most confident subset \mathcal{D}_{conf} of the generated *pseudo-label* data $\mathcal{D}_{PL} = \{(\mathbf{x}_i, \tilde{y}_i)_{i=1}^m\}$ as follows

$$\mathcal{D}_{conf} = \{(\mathbf{x}, \tilde{y}) \in \mathcal{D}_{PL} \mid (s(\theta, \mathbf{x}) < \tau_n) \vee (s(\theta, \mathbf{x}) > \tau_p)\}. \quad (3.24)$$

Thus, we select only those observation-label tuples that the classifier confidently deems negative or positive. More formally, we select only the tails of the confidence distribution over the *pseudo-label* dataset \mathcal{D}_{PL} . Rizve et al. [Riz+21] show that a decrease in prediction uncertainty correlates with a decrease in the *expected calibration error* [Guo+17]

$$\mathbb{E}_{\hat{P}} [|\mathbb{P}(\hat{Y} = Y | \hat{P} = p) - p|] \quad (3.25)$$

with \hat{P} , the confidence estimate or probability of correctness, and \hat{Y} , the predicted class. For a perfectly calibrated model the confidence estimate \hat{P} should be equal to the actual class probability p .

However, the selection of a *pseudo-label* data subset associated with confident predictions might still lead to the inclusion of wrong predictions also known as *noisy labels* caused by overconfident, miscalibrated models. Though, Niculescu et al. find that in contrast to neural networks in multi-class settings, neural networks for binary classification tend to predict unbiased and well calibrated probabilities [NMC05].

Label Noise and Early Stopping

Generated *pseudo-label* data as well as data annotated by human experts might contain *noisy labels*. These are falsely assigned labels which can be a result of human error due to miscommunication, unclearly defined decision rules or insufficient domain knowledge on the human side. *Pseudo labels* might be noisy because the model generating the labels is underfitted or it might be that the model is overfitted and overconfident. Zhu et al. investigated the impact of noisy labels on the training of LLMs, specifically BERT, and showed that the fine-tuning with noisy labeled data first leads to a good generalization performance that later degrades when the model begins to overfit on the noisy labels [Zhu+22]. They propose the employment of *early stopping* to avoid the overfitting on noisy labeled data. Additionally, Tanzer et al. find that LLMs show a very low susceptibility to *label noise* and overfitting in general. They analogously propose an early stopping procedure to mitigate the effects of memorizing label noise [TRR22].

To detect overfitting and label noisy memorization, we utilize a combination of cross validation, threshold-based checkpointing and an additional validation dataset consisting of difficult samples and a slightly different distribution where the concrete implementation of this procedure will be explained in Chapter 5.

3.2.3 *Label Imbalance*

Many classification problems have to deal with severely imbalanced data, since the underlying population is highly imbalanced. For example, a detector assigning the classes *normal* and *suspicious activity* to network logs of an enterprise network or a classifier trying to decide if there’s cancerous growth present in a medical image. In both cases we would expect the default cases or negative classes, *normal behaviour* and *non-cancerous*, to be much higher in number than the positive classes.

The evaluation of such a classifier with the *accuracy* measure or a loss function implicitly optimizing the *accuracy* measure might look promising at first, but, depending on the severity of the label imbalance and thus the sparsity of the positive class, the performance on the positive class might be sub-par, which manifests in low *precision* and *recall* measures. This is particularly problematic in cases where the positive class is of much higher interest than the negative class.

As mentioned in Section 3.2.1, the annotation of data is often expensive and time-consuming, which leads to labeled data being only sparsely available. In Section 3.1.2 we have seen that the problem of data and especially label sparsity is amplified in multi-label classification linearly up to exponentially in \mathcal{L} , depending on the multi-label problem modeling strategy.

Different methods to handle label imbalance have been proposed, which can be divided into methods like *over-* and *undersampling*, modifying the distribution of the training data, and methods modifying the statistical model or training procedure, for example through a class weighted loss function.

As discussed in [Section 3.1.1](#), we aim to train a classifier that learns the joint conditional probability distribution $\mathbf{p}_x(\mathbf{Y})$, which we assume to be the product of the independent conditional marginal probability distributions $\mathbf{p}_x(Y_i)$. Applying *Bayes' Theorem*, we can formalize finding the correct class y^* in a binary classification task as such

$$\begin{aligned} y^* &= \arg \max_{c \in \{0,1\}} p(Y = c | \mathbf{X} = \mathbf{x}) \\ &= \arg \max_{c \in \{0,1\}} \frac{p(\mathbf{X} = \mathbf{x} | Y = c) p(Y = c)}{p(\mathbf{X} = \mathbf{x})} \\ &= \arg \max_{c \in \{0,1\}} p(\mathbf{X} = \mathbf{x} | Y = c) p(Y = c). \end{aligned} \tag{3.26}$$

We select the class c that maximizes the product of the likelihood $p(\mathbf{X} = \mathbf{x} | Y = c)$ and the class prior $p(Y = c)$.

Resampling

Resampling techniques, such as majority class oversampling and minority class undersampling, have a direct impact on both of these quantities. Originally, we assumed our training data to be an *i.i.d.* sample taken from an unknown distribution $p(\mathbf{X}, Y)$ with class prior $p(Y)$. By undersampling the majority class, or oversampling the minority class, we change the class frequencies in our training data and thus the empirical class prior distribution $p_{emp}(Y)$. As such the assumption, that our training data is an *i.i.d.* sample does not hold anymore. The effect of this deviation (on the decision boundary) can be seen in [Figure 3.4](#).

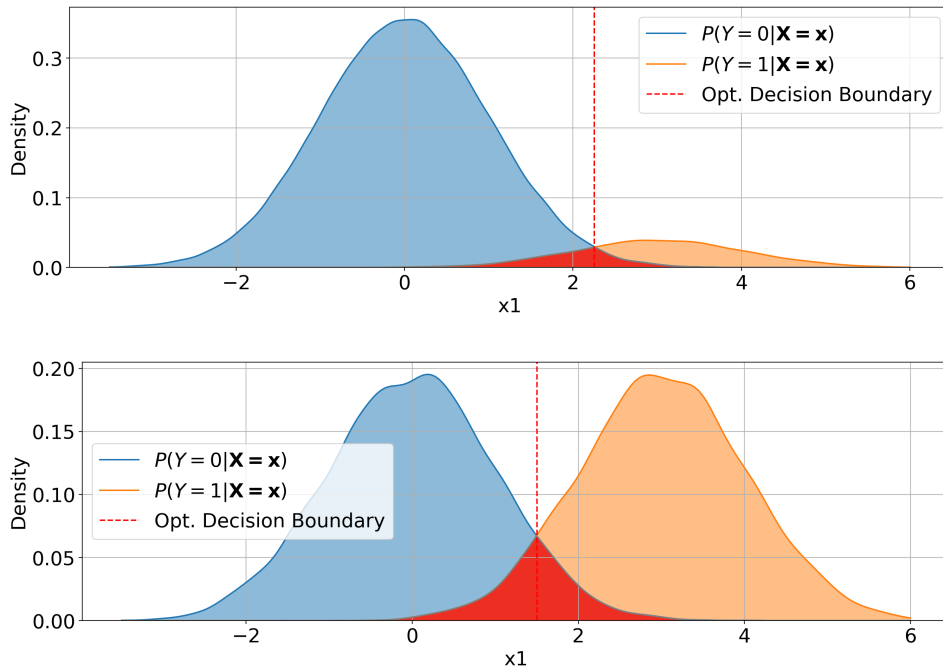


Figure 3.4: Effect of majority class undersampling on the optimal decision boundary with i.i.d. sample (top) and majority class undersampling (bottom)

So, why would we consider knowingly violating this assumption? If we are confronted with severe class imbalance, for example a positive-negative ratio of 1:500, we'd average 2 positive examples in an annotated training dataset of 1000 examples. As such, it would be impossible for a model to learn any meaningful structure of the data associated with the positive class. With respect to Equation 3.26 this means, that the model could possibly only learn a very crude approximation of the likelihood of the positive class, $p(\mathbf{X} = \mathbf{x}|Y = 1)$. Given the aforementioned cost and time constraints with respect to data annotation, a slight deviation from the true class prior might be worthwhile, if we can thereby improve the diversity of observations associated with the minority class.

Instead of or in conjunction with changing the distribution of the available data through resampling techniques, we can also try to increase the models attention to the minority class by assigning weights to the classes. We show how a loss function can be modified to account for label imbalance in Section 3.3.5. Another way to account for the label imbalance without necessarily changing the training data distribution, is to tune the decision threshold, which is used to decide between negative and positive prediction.

Threshold tuning

As explained in Section 3.1.3, the F1-score serves as a balanced measure between detecting relevant samples (recall) and being correct when predicting

positives (precision). An often employed decision threshold is $t = 0.5$, the midpoint between the probability minimum and maximum. Though, this threshold is not necessarily the optimal threshold with respect to the F1-score. When training the model with an unweighted binary cross-entropy loss or focal loss under severe class imbalance ie. typically a low number of positives, the model might learn to pay less attention to the positive class for its less prominent influence on the overall loss. This in turn leads to predicted probabilities skewed to lower values. Different methods for the optimization of the decision thresholds in a multi-label setting have been proposed. These methods generally are split into global and local methods and depend on the label (in-)dependence assumptions and multi-label modeling strategy employed [Yano1]. As explained in Section 3.1.2, the BR approach to multi-label classification assumes conditional label independence. This circumstance is exploited in the local *sCut* threshold optimization method, where the decision threshold for each class is optimized individually. For this a validation data set is used and the best decision threshold for each class is determined by performing a linesearch over candidate decision thresholds $t_i \in (0, 1) \subset \mathbb{R}$.

3.3 ATTENTION-BASED DEEP NEURAL NETWORKS

The introduction of the Transformer neural network architecture by Vaswani et al. [Vas+17] in 2017 had significant impact on the Natural Language Processing (NLP) landscape. A quantification of the impact the *Transformer* architecture had on various NLP tasks can be seen in the leaderboard¹ of the popular General Language Understanding Evaluation (GLUE) benchmark [Wan+18], which is as of the time of this writing dominated by derivations and successors of the *Transformer* architecture. The GLUE benchmark tries to cover a broad spectrum of Natural Language Understanding (NLU) tasks and consists of nine tasks such as *Question Answering*, *Named Entity Recognition* and *Sentiment Analysis*. Multiple factors contribute to the state-of-the-art performance and success of the Transformer and its successors. Though, two important parts are played by the replacement of sequential network components with the *Attention* mechanism presented by Bahdanau et al. [BCB14] in 2014 and the development of *Pre-Training* schemes enabling transfer learning with *pre-trained language models*.

In this section we give an overview of the components of the *Transformer* architecture. We focus on the *Encoder* part of the architecture that is used to generate outputs for the binary and multi-label classification tasks at hand. To utilize deep learning models in context of NLP, the textual data has to be transformed into a suitable numeric representation. Consequently, we present an overview of different text representation techniques in Section 3.3.1. In Section 3.3.2 we try to provide a holistic view of the *Transformer Encoder Architecture*. Following, we discuss the *Attention* mechanism and its advantages over sequential mechanisms for context learning in Section 3.3.3. We conclude this section by looking at the *Pre-Training* scheme proposed by Devlin et al. in 2018 [Dev+18] and extended by Clark et al. in 2020 [Cla+20] enabling efficient transfer learning with LLMs in Section 3.3.4.

3.3.1 Text representation

Digital textual data is generally represented as a string of characters with a text-encoding. The encoding specifies which character or symbol is represented by a specific single- or multi-byte sequence. To use natural language in context of a machine learning or specifically neural network model, we have to find a suitable numeric representation for the textual data. The collection of textual documents contained in our training data is called a *text corpus*. Historically there have been a multitude of approaches to (pre-)process and transform textual data into such a numerical representation. The collective starting point for many of these approaches is called *tokenization* and encompasses the transformation of documents into a list of tokens with the tokens being (sub-)words or multi-word combinations known as *n-grams*. Additionally, the process of tokenization might be accompanied by remov-

¹ <https://gluebenchmark.com/leaderboard>, (Screenshot in the appendix, Figure A.1)

ing *stopwords*, *lemmatization* and the replacement of *synonyms*. Jurafsky et al. provide an extensive overview of text preprocessing techniques in [JM09].

The tokenization of the documents in a training corpus yields a list of tokens for each document as well as a corpus-specific vocabulary V . The vocabulary is a dictionary containing the key-value mappings between each unique token and its corresponding numeric identifier or index. We can formalize the vocabulary mapping as

$$\begin{aligned} V : t &\rightarrow i \\ V^{-1} : i &\rightarrow t \end{aligned} \tag{3.27}$$

where the vocabulary V maps a token t to a corresponding ID i and the inverse of that mapping V^{-1} maps an ID i to a token t . As such, a tokenized document can then be represented as a collection of integer IDs. Proceeding from here, there are multiple ways to create numerical vector or tensor representations of the tokenized documents. The representation of a token, word or document in a vector space \mathcal{X} is known as word, token or document *embedding*, respectively. In the following, we provide a brief summary for different *encoding* and *embedding* approaches:

ONE-HOT ENCODING: This is a very simple method of representing each token t of vocabulary V in a vector space $\mathcal{X} = \{0, 1\}^{|V|}$. Each token t is represented as a vector of dimension $|V|$, where each vector element is zero except for the element at index $i = V(t)$, which is one.

COUNT-BASED METHODS: Document vectors based on word counts can be understood as the sum of one-hot vectors of all the tokens in a given document. As such, a dense document vector c based on word counts is of dimension $|V|$ with an element c_i being the count of token $t = V^{-1}(i)$ in the given document. To alleviate the impact of different document lengths, it is also possible to use normalized document vectors

$$c_{normalized} = \frac{c}{|c|} \tag{3.28}$$

TFIDF: The Term Frequency Inverse Document Frequency (**TFIDF**) approach ([RU11], p. 8) combines document local, *term frequency*, and global statistics, *inverse document frequency*, to create a numeric document vector. The **TFIDF** score can be viewed as the importance a token t has for a given document D and is calculated as

$$TFIDF(t, D) = tf(t, D) \cdot idf(t) \tag{3.29}$$

with the *term frequency* being local to document D and the *inverse document frequency* being a global measure over all documents in the corpus. As such, the **TFIDF** score of a token is especially high, if the token appears frequently in the current document, but appears infrequently in the corpus of all documents.

WORD2VEC: Word2Vec [Mik+13] is an umbrella term for two different approaches utilizing a two-layer neural network and a context window of fixed size to create a vector space (*word embeddings*) from a text corpus. The two methods are called *Continuous Bag of Words (CBOW)* and *Skip-Gram*, with both methods slightly differing in architecture and training procedure. In the **CBOW** method, the neural network learns to predict a center word w_t from the surrounding context of size c . The corresponding objective function therefore tries to maximize the conditional probability of token or word w_t given its surrounding context

$$\mathcal{L} = \frac{1}{T} \sum_t \log p(w_t | w_{t-\frac{c}{2}}, \dots, w_{t-1}, w_{t+1}, \dots, w_{t+\frac{c}{2}}). \quad (3.30)$$

The neural network in the *Skip-Gram* approach on the other hand tries to predict the surrounding context of a word w_t . This objective is optimized by maximizing the sum of the log probabilities of all context words surrounding word w_t

$$\mathcal{L} = \frac{1}{T} \sum_t \sum_{-\frac{c}{2} \leq j \leq \frac{c}{2}, j \neq 0} \log p(w_{t+j} | w_t). \quad (3.31)$$

The word embeddings obtained through the *Word2Vec* approaches incorporate a global average of context information. Each time a word w appears in the training corpus its surrounding context will contribute to the adjustment of the neural networks parameters and thus to the final word embeddings of the given word w . Unfortunately, the meaning of a word is often highly dependent on its local context and might vary by a lot. By representing a word with a fixed vector, the embedding will reflect a global context average dominated by the most common context of a word.

DEEP CONTEXTUALIZED WORD REPRESENTATIONS: To capture syntax, semantics as well as local linguistic context in word representations, methods utilizing more complex deep learning architectures have been researched as mentioned in [Chapter 2](#). One of these architectures is the *Transformer* and the *BERT* pre-training scheme, which both will be laid out in detail in the following sections.

3.3.2 Transformer Encoder Architecture

The *Transformer* consists of two main building blocks, which are the *Encoder* and *Decoder*. The original model was proposed mainly *Machine Translation* in mind, which at that point in time was typically approached with *Encoder-Decoder* models utilizing convolutional neural networks (**CNNs**) and **RNNs** to capture and represent contextual information contained in the sequential inputs [SVL14][KB13]. The *Transformer* model however replaces the sequential architecture components with the *Attention* mechanism. The general idea

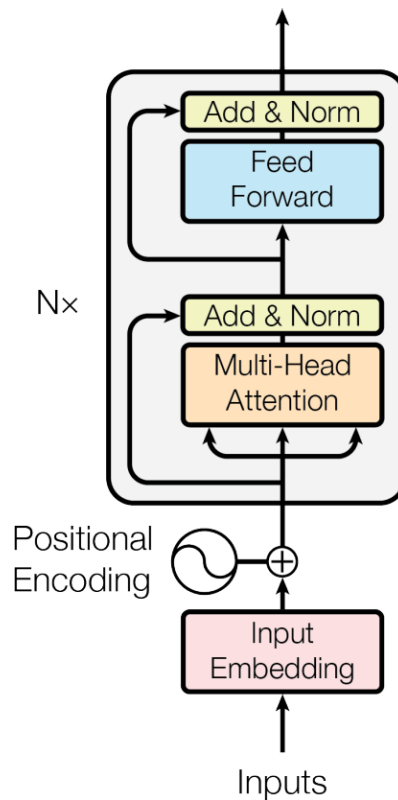


Figure 3.5: Transformer Encoder Architecture [Vas+17]

of the *Encoder-Decoder* architecture is to first use the *Encoder* to transform a sequential input into an internal contextualized representation and then use the *Decoder* to transform the internal representation into the task-specific output. For example, in a *Machine Translation* task the sequential input might be a German sentence that is to be translated into a French output sentence. Another example is *Question Answering*, where the sequential input is a question and the output of the decoder is the corresponding answer. In context of *MLC*, the *Decoder* part of the architecture is not required, since we can directly utilize the internal contextualized representations output by the *Encoder*.

Here we will provide an overview of the *Transformer Encoder* architecture with specifically referencing the implementation of Bidirectional Encoder Representations from Transformers (*BERT*) [Dev+18] in mind. In Figure 3.5 an illustration of the high-level architecture of the *Encoder* by Vaswani et al. [Vas+17] can be seen.

Input Embedding

The tokenized inputs are first mapped into an embedding space with dimension $\mathbb{R}^{|\mathcal{V}| \times d_h}$. Thus, an embedded token will be represented by a vector of dimension d_{model} . This dimension is kept constant throughout the layers of the Transformer. The embeddings are generally just a matrix of trainable

parameters and a dictionary mapping an input ID to a specific row in the matrix.

Positional Encoding

Since the *Transformer* utilizes the *Attention* mechanism instead of sequential components like *RNNs*, an additional mechanism is needed to provide the model with the sequential information of the input data. This is done by adding a positional encoding to the embedded inputs. An *absolute positional encoding* is calculated with the following functions

$$\begin{aligned} PE(pos, 2i) &= \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right) \\ PE(pos, 2i + 1) &= \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right) \end{aligned} \quad (3.32)$$

where pos is the position in the sequence and i is the index within the hidden dimension d_{model} . Vaswani et al. hypothesize that the encoding enables the model to attend to relative positions within the input sequence.

Encoder Blocks

The grey block in [Figure 3.5](#) is one of N *Encoder* blocks that are stacked on top of each other. On a high level it is composed of a *Multi-Head Attention* block and a *feedforward neural network (FNN)* part. We will set aside the explanation of the *Multi-Head Attention* block until the following [Section 3.3.3](#). The *Multi-Head Attention* and the *FNN* block are followed by a *Residual Connection* [[He+15](#)] and a *Layer Normalization* layer [[Xu+19](#)]. Both, *Residual Connections* and *Layer Normalization* enable a faster and more stable training of the neural network. *Layer Normalization* does this by shifting and rescaling the intermediary output of the preceding layer and as such mitigates the problems of *dying* and *exploding* gradients. *Residual Connections* on the other hand enable a neural network layer to learn a *zero-mapping* instead of an *identity-mapping* if the layer, informally spoken, finds nothing valuable to learn. The *zero-mapping* is generally much easier to learn, ie. just set every parameter close to zero, than the *identity-mapping* [[He+15](#)]. If the part of the network enclosed by a *Residual Connection* has difficulties learning, the gradient can still freely flow back through the *Residual* or *Skip-Connection*, which mitigates the *dying gradient* problem.

In [Figure 3.6](#) a more detailed visualization of an *Encoder* block by Alamar et al. [[Ala18](#)] is shown. Here an exemplary sentence ‘*Thinking Machines*’ is processed that has been tokenized and mapped into the embedding space \mathcal{X} as matrix X with the word embeddings as row vectors x_1 and x_2 . Before being input to the **first** *Encoder* block, the *positional encoding* is added to the embeddings. For didactic reasons the processing of both word embeddings

is seen in an unrolled fashion where in reality this is performed through combined matrix operations.

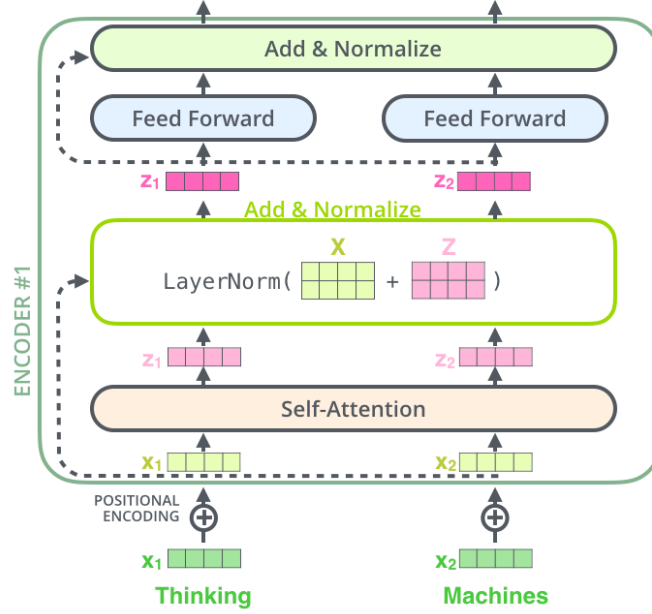


Figure 3.6: Transformer Encoder Block [Ala18]

The *Self-Attention* layer outputs contextualized representations z_1 and z_2 which are then added to the positionally encoded inputs x_1, x_2 with the help of a *Residual Connection*. The resulting sum of both matrices is normalized with the *Layer Normalization* layer. The *Residual Connections* can also be understood as a pass-through for information of earlier layers. The second part of the *Encoder* block functions analogously, but uses a *FNN* block instead of a *Self-Attention* layer to transform the normalized intermediary outputs z_1 and z_2 . The *feed forward* block is a fully connected neural network performing an affine transformation to the inputs

$$\begin{aligned} z &= f_{aff}(x) \\ &= x\mathbf{W}^T + b. \end{aligned} \tag{3.33}$$

The weight matrix \mathbf{W} and the bias term b are of size $\mathbf{W} \in \mathbb{R}^{4d_{model} \times d_{model}}$ and $b \in \mathbb{R}^{4d_{model}}$. Thus, the inputs are mapped into a vector space four times its original dimension. After the affine transformation, a subsequent Gaussian Error Linear Unit (*GELU*) non-linear activation function is applied to the outputs of the *FNN*

$$a = GELU(z) \tag{3.34}$$

$$= 0.5z \left(1 + \tanh \left[\sqrt{2/\pi} (z + 0.044715z^3) \right] \right). \tag{3.35}$$

This enables the network to learn complex, non-linear relationships. The final *Add & Normalize* block consists of a *FNN* layer, a *Dropout* layer [Sri+14]

and *Layer Normalization*. The weight matrix \mathbf{W} of the final **FNN** is chosen in such a way that it returns outputs with the original model dimensions, d_{model} , and as such the weight matrix is $\mathbf{W} \in \mathbb{R}^{d_{model} \times 4d_{model}}$.

3.3.3 Attention

The arguably most important innovation of the *Transformer* architecture is the usage of the *Attention* mechanism instead of sequential mechanisms like **RNNs** to create contextualized intermediary representations of the input data.

To provide a point of reference and motivation for the *Attention* mechanism, we first take a short detour to look at how we can utilize context information with **RNNs**. An unrolled unidirectional **RNN** can be seen in **Figure 3.7** processing a sequential input $\mathbf{x} = (x_1, \dots, x_t)$ and outputting a contextualized *hidden state* $\mathbf{h} = (h_1, \dots, h_t)$.

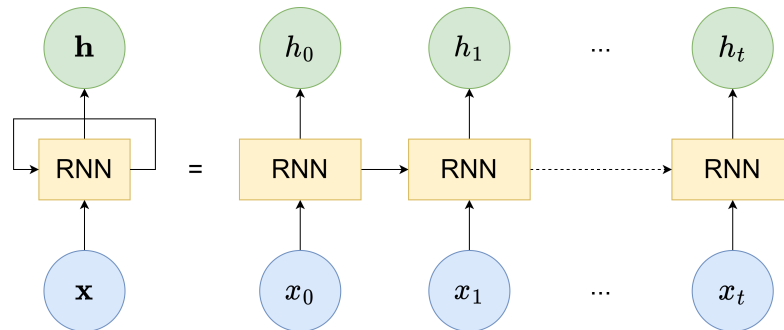


Figure 3.7: Unrolled **RNN**

Each hidden state h_i is a combination of the corresponding input token x_i and the previous hidden state h_{i-1} with this recursive structure unwrapping until the first input token x_0 . Hence, only the final hidden state h_t has access to all the contextual information contained in the input sequence and is used as the output of the **RNN**. Additionally, important contextual information found at the beginning of the sequence has to traverse the complete sequential structure and might degrade in that process. At the same time, if very important contextual information appears at the end of the sequence, the hidden states of the previous time steps aren't able to utilize it.

Instead of such a recursive sequential structure with the aforementioned disadvantages, *Attention* utilizes a highly parallel structure enabling each token to attend to any other token. We distinguish between *Self-Attention*, where the tokens of the input sequence attend to each other, and *Cross-Attention*, handling two differing token sequences. The latter appears in the *Encoder-Decoder* architecture and is needed in tasks like translation. An exemplary visualization of the attention scores resulting from *Self-Attention* and *Cross-Attention* is shown in **Figure 3.8**. In both cases the token sequence on the left side is attending to the token sequence on the right side. The line thickness

as well as the strength of the background color of the tokens on the left side indicate how strongly the specific token on the left side attends to the selected token on the right side.

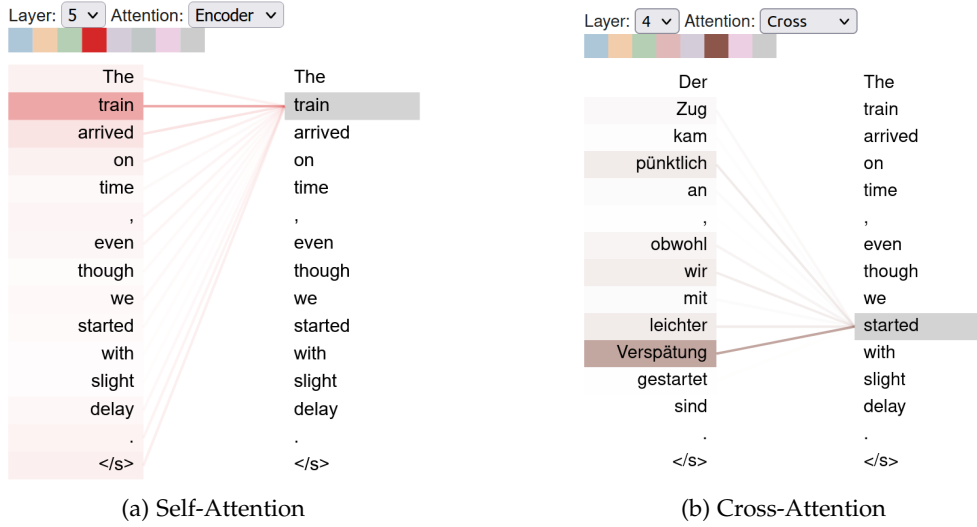


Figure 3.8: Self-Attention and Cross-Attention visualized with BertViz [Vig19]

In practice, the attention scores $\alpha_{i,j}$ visualized by line thickness are contained in an attention matrix A , which in case of *Self-Attention* is a quadratic matrix $\mathbf{A} \in \mathbb{R}^{n_{seq} \times n_{seq}}$ and a rectangular matrix $\mathbf{A} \in \mathbb{R}^{n_{seq} \times n_{seq2}}$ in case of *Cross-Attention*. There are different ways to calculate the attention scores like the *Bahndanau Attention* [BCB14] using a neural network or the *Scaled Dot-Product Attention* mechanism [Vas+17] proposed by Vaswani et al. We focus on the latter, since this is the mechanism employed by BERT and its descendants ELECTRA and DistilBERT, which we will discuss in Section 3.3.4 and Section 4.2, respectively.

Scaled Dot-Product Attention

Attention can be understood as a mechanism analogous to an information retrieval process with input queries $\mathbf{Q} \in \mathbb{R}^{d_{model} \times d_k}$ and a set of key-value pairs, with the keys, $\mathbf{K} \in \mathbb{R}^{d_{model} \times d_k}$, and the values, $\mathbf{V} \in \mathbb{R}^{d_{model} \times d_v}$. We can compare the process with the usage of a search engine. We provide a query to the search engine and expect it to return a ranking of the best fitting results from a database. Instead of returning ranked results, the attention mechanism returns a weighted average of the values corresponding to the best matching keys:

$$\begin{aligned}
 z_i &= \text{Attention}(q_i, \mathbf{K}, \mathbf{V}) \\
 &= A_{i,:} \mathbf{V} \\
 &= \sum_{j=1}^{n_{seq}} \alpha_{i,j} V_j.
 \end{aligned} \tag{3.36}$$

Formally *Scaled Dot-Product Attention* is defined as follows

$$\begin{aligned} \mathbf{Z} &= \text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) \\ &= \text{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right) \mathbf{V} \\ &= \mathbf{A}\mathbf{V}. \end{aligned} \quad (3.37)$$

where the *Softmax*-function is applied row-wise and defined as

$$\text{Softmax}(\vec{x})_i = \frac{\exp x_i}{\sum_{j=1}^n \exp x_j} \quad (3.38)$$

for row-vector element x_i .

To get a good understanding of the *Self-Attention* mechanism, we'll explain the individual steps by example of an input sequence \mathbf{x} , where we want to calculate the self-attention scores corresponding to token x_i . A visualization of the necessary steps by Vaswani et al. [Vas+17] is shown on the left in [Figure 3.9](#).

1. **INPUTS TO QUERY, KEY AND VALUE:** The input sequence \mathbf{x} is linear transformed into the query $\mathbf{Q} \in \mathbb{R}^{d_{\text{seq}} \times d_k}$, key $\mathbf{K} \in \mathbb{R}^{d_{\text{seq}} \times d_k}$ and value matrices $\mathbf{V} \in \mathbb{R}^{d_{\text{seq}} \times d_v}$ by applying the weight matrices $\mathbf{W}^{\mathbf{Q}} \in \mathbb{R}^{d_h \times d_k}$, $\mathbf{W}^{\mathbf{K}} \in \mathbb{R}^{d_h \times d_k}$ and $\mathbf{W}^{\mathbf{V}} \in \mathbb{R}^{d_h \times d_v}$ respectively. The weight matrices consist of parameters learned during the training of the network. For one token of the sequence we have

$$\begin{aligned} q_i &= x_i \mathbf{W}^{\mathbf{Q}} \\ k_i &= x_i \mathbf{W}^{\mathbf{K}} \\ v_i &= x_i \mathbf{W}^{\mathbf{V}}. \end{aligned} \quad (3.39)$$

Self-Attention and Cross-Attention

In *Self-Attention*, the query \mathbf{Q} , key \mathbf{K} and value \mathbf{V} matrices are linear transformations of the same input sequence \mathbf{x} , since we want to learn contextual relationships within the same sequence. However, in *Cross-Attention*, utilized in a *Transformer Encoder-Decoder architecture*, the key \mathbf{K} and value \mathbf{V} matrices originate from the output of the *Encoder* and the query \mathbf{Q} matrix is a linear projection of the input sequence to the *Decoder*. Hence, there are two different sequences at play, which both are mapped into the same vector space and as such their dot-product can be calculated.

2. **DOT-PRODUCT SIMILARITY:** The raw dot-product similarity $s_{i,j}$ between the query \mathbf{q}_i and each key \mathbf{k}_j is calculated as

$$s_{i,j} = q_i k_j \quad (3.40)$$

and for the full sequence, the similarity matrix \mathbf{S} with raw dot-product similarity scores is calculated as

$$\mathbf{S} = \mathbf{Q}\mathbf{K}^T. \quad (3.41)$$

3. **SCALING THE DOT-PRODUCT:** Vaswani et al. [Vas+17] find that in case of a large key vector space dimension d_k , the dot-products tend to become very large, which leads to problematic *Softmax* gradient values. Hence, they propose a scaling factor as the inverse of the square root of d_k

$$s'_{i,j} = \frac{s_{i,j}}{\sqrt{d_k}}. \quad (3.42)$$

4. **MASKING (OPTIONAL):** In this optional step a large negative value is added to the scores $s'_{i,j}$ for sequence positions that we do not want to attend to. Applying a large negative value leads to *Softmax* scores close to zero, which means that we do not attend to this positions. This is necessary if we use tensors of the same length for the sequence dimension. Sequences shorter than the fixed maximum length could potentially attend to the empty positions at the end.

5. **APPLYING THE SOFTMAX FUNCTION:** In this step each row of the scaled score matrix \mathbf{S}' is transformed into a probability distribution by application of the Softmax function

$$A_{i,:} = \text{Softmax}(\mathbf{S}'_{i,:}). \quad (3.43)$$

Hence, the attention scores or attention weights $\alpha_{i,j}$ of a row i sum to one

$$\sum_{j=1}^{n_{seq}} \alpha_{i,j} = 1. \quad (3.44)$$

6. **WEIGHTED VALUE AVERAGE:** The attention scores or attention weights are now used to create a weighted average of the values. Thus, values corresponding to keys that were strongly attended to will be more prominently represented in the output vector.

$$z_i = A_{i,:}\mathbf{V} \quad (3.45)$$

7. **OUTPUT LINEAR TRANSFORMATION:** As mentioned at the beginning of the section, the hidden dimension d_{model} is kept consistent throughout the *Encoder* layers. As such a final linear transformation is applied to the weighted average of values $\mathbf{Z} \in \mathbb{R}^{n_{seq} \times d_v}$ returned from (6). This is done by multiplying each of the vectors z_i with the learnable output weight matrix $\mathbf{W}^O \in \mathbb{R}^{d_v \times d_{model}}$ as such

$$z'_i = z_i \mathbf{W}^O \quad (3.46)$$

and thus projecting each z_i back into the original model space $\mathbb{R}^{d_{model}}$.

Multi-Head Attention

To enable the learning of more diverse positional and linguistic relationships Vaswani et al. propose *Multi-Head Attention*, where the query, key and value are projected h times into smaller representation subspaces $\mathbf{QW}_i^Q, \mathbf{KW}_i^K$ and \mathbf{VW}_i^V with $i \in \{1, \dots, h\}$. Each of these perform the attention steps layed out in the previous subsection in parallel. The modified architecture can be seen in the visualization by Vaswani et al. [Vas+17] on the right side of Figure 3.9.

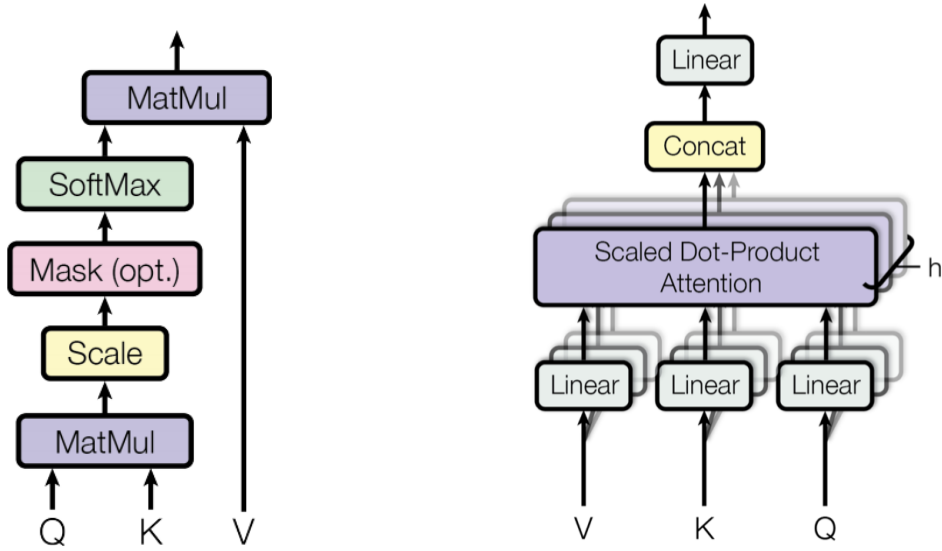


Figure 3.9: Scaled Dot-Product Attention (left) and Multi-Head Attention (right) [Vas+17]

To avoid prohibitive memory and computation requirements, the dimension of the query, key and value spaces in *Multi-Head Attention* is modified by a scaling factor inversely proportional to the number of heads

$$d_k = d_v = d_{model}/h.$$

For each head we have

$$\mathbf{Z}_i = \text{Attention}(\mathbf{QW}_i^Q, \mathbf{KW}_i^K, \mathbf{VW}_i^V) \tag{3.47}$$

with $\mathbf{Z}_i \in \mathbb{R}^{n_{seq} \times d_v}$ and $d_v = d_{model}/h$. By concatenating the h heads along the hidden dimension axis, we recover the dimension of the original single-head formulation

$$\mathbf{Z} = [\mathbf{Z}_1 \cdot \dots \cdot \mathbf{Z}_h]. \tag{3.48}$$

The final output of the *Multi-Head Attention* layer is obtained by linear transforming \mathbf{Z} with the output matrix $\mathbf{W}^O \in \mathbb{R}^{hd_v \times d_{model}}$ analogous to the single-head formulation

$$\mathbf{Z}' = \mathbf{ZW}^O. \tag{3.49}$$

Each attention head is able to attend to information from an individual representation subspace or, stating this rather informally, each attention head is able to provide an individual perspective on the input sequence. To show this by example, [Figure 3.10](#) displays the attention scores of different heads of the same *Encoder* layer. The heads attention distributions reflect different positional and linguistic patterns. The first head attends specifically to the next position in the sequence whereas the second heads attention distribution is more shallow and attends to most of the tokens in the sequence. The third heads attention pattern on the other hand seems to be much more related to the content of the sequence and could be interpreted as *time* context.

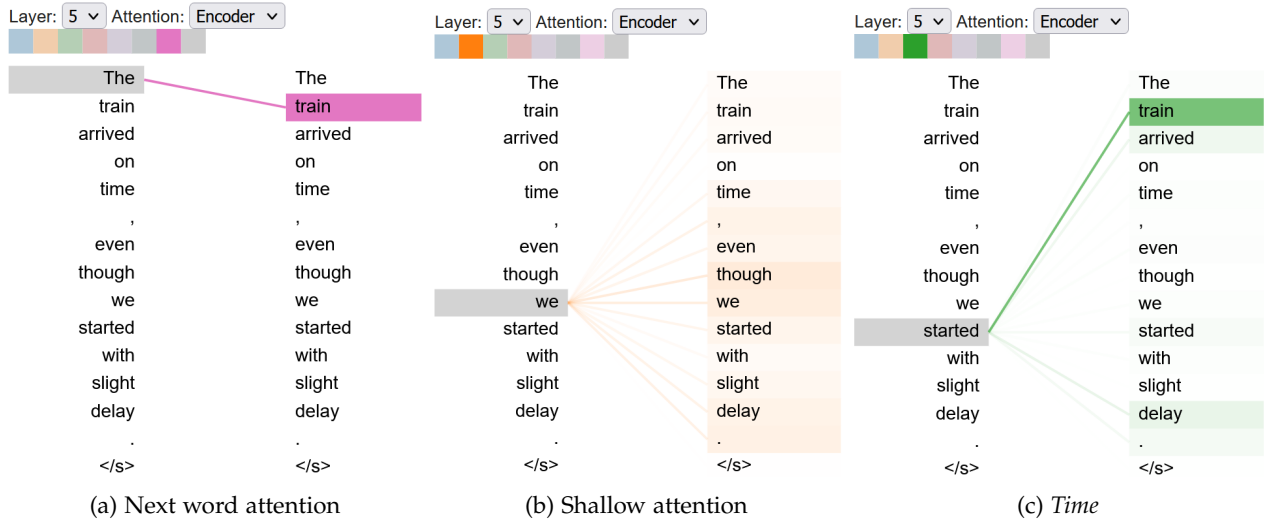


Figure 3.10: Different Self-Attention Heads visualized with *BertViz* [[Vig19](#)]

An extensive investigation into patterns exhibited by the attention heads of pre-trained [BERT](#) models has been provided by Clark et al. [[Cla+19](#)]. They distinguish between *Surface-Level* patterns and linguistic patterns. *Surface-Level* patterns are for example positional patterns the attention is focused on the previous or next token as shown in [Figure 3.10](#) (left). Another form of *Surface-Level* pattern is the attention solely on special tokens. Linguistic patterns are for example verbs attending to a corresponding object or possessive pronouns attending to nouns.

3.3.4 Pre-Training and Fine-Tuning

The high learning capacity of large language models can be attributed to the utilized network architectures and especially to the large number of learnable parameters contained in these models. In [Table 3.3](#) the number of learnable parameters contained in a selection of [LLM](#) architectures is shown.

| Model name | Number of parameters |
|------------|----------------------|
| DistilBERT | 66M |
| BERT base | 110M |
| BERT large | 314M |
| GPT-3 | 175B |

Table 3.3: Parameter count of Pre-Trained Transformer models

Training a model with millions of parameters from scratch requires large amounts of training data and computational resources. Fortunately, *LLMs* generalize very well, which makes pre-trained *LLMs* highly suitable for transfer learning by a relatively inexpensive, subsequent fine-tuning. Brown et al. [Bro+20] show that very large models, specifically GPT-3, even outperform a fine-tuned BERT large in the *SuperGLUE* benchmark [Wan+19], an extended collection of *NLU* tasks, by providing it with only a few labeled examples, also known as *Few-Shot Learning*.

The pre-training of *BERT* consists of two unsupervised learning tasks, which are *Masked Language Modeling* and *Next Sentence Prediction*.

MASKED LANGUAGE MODELING: In this task 15% of the tokens in the training data are randomly replaced with a special [MASK] token. The model is trained to predict the correct original word at the masked position.

NEXT SENTENCE PREDICTION: For the *Next Sentence Prediction* task the training data is first split sentence-wise and transformed to inputs consisting of adjacent sentence pairs. For 50% of the sentence pairs the succeeding sentence *B* is replaced by a random sentence taken from the training corpus. The training objective is to correctly predict, if sentence *B* is the original successor to sentence *A*.

The inputs of *BERT* are an additive combination of *token embeddings*, *position embeddings* and *segment embeddings* as shown in Figure 3.11.

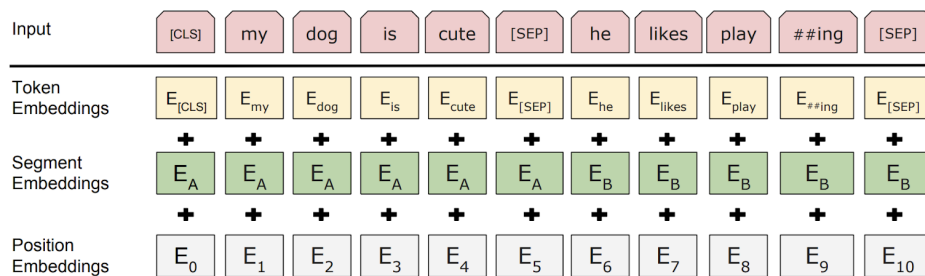


Figure 3.11: Inputs used *BERT*

The *token* and *position embeddings* are as explained in Section 3.3.2. The *segment embeddings* serve to encode if a token belongs to sentence *A* or *B*.

ELECTRA: Modified pre-training scheme

Efficiently Learning an Encoder that Classifies Token Replacements Accurately (ELECTRA) is a modified *Masked Language Modeling* pre-training scheme [Cla+20]. Instead of masking tokens at random with fixed probability, *ELECTRA* uses a Generative Adversarial Network (GAN) approach [Goo+14] to replace tokens with a *slightly wrong* token. The discriminator has to decide, if the token is the original or a replaced token.

3.3.5 *Multi-Label Classification with Transformers*

To perform multi-label classification with BERT or related LLMs the model architecture has to be extended by a *Multi-Label Classification Head*. This building block projects the hidden state corresponding to BERTs [CLS] token into an output space with dimension $d_{|\mathcal{L}|}$ through the application of a FNN layer

$$\mathbf{y} = h_{[CLS]} \mathbf{W} + b \quad (3.50)$$

with the resulting logits $\mathbf{y} \in \mathbb{R}^{|\mathcal{L}|}$, the output matrix $\mathbf{W} \in \mathbb{R}^{d_{model} \times |\mathcal{L}|}$ and the bias term $b \in \mathbb{R}^{|\mathcal{L}|}$. To arrive at the desired probability output, we apply the *sigmoid* or *logistic* function to each logit y_i with $i \in \{1, \dots, |\mathcal{L}|\}$ individually

$$\begin{aligned} \hat{p}_i &= \sigma(y_i) \\ &= \frac{1}{1 + e^{-y_i}}. \end{aligned} \quad (3.51)$$

This is in agreement with the BR approach to modeling the MLC problem at hand, since we are not explicitly modeling any possible label dependencies. Yet, the fine-tuned model is able to implicitly make use of labels correlations by accessing the same feature space for the creation of each binary output.

Loss functions

There are different objective functions to measure the quality of the probability outputs produced by our multi-label model. The choice of objective or *loss* function depends on the multi-label modeling approach as discussed in Figure 3.2 as well as the distribution or balance of labels. Given the binary relevance approach to the MLC problem at hand and the existence of label imbalance in the data, we focus on two specific loss functions and their corresponding parameterization. These are *Binary Cross-Entropy Loss* and *Focal Loss*.

BINARY CROSS-ENTROPY LOSS: The Binary Cross-Entropy (BCE) loss function shows desirable mathematical properties in that it is smooth and differentiable. It serves as a surrogate loss for the *Hamming Loss* and is well-suited for binary relevance multi-label classification. The BCE loss

for multi-label classification is defined as the sum of the [BCE](#) losses for each individual binary classification problem and defined as follows

$$\mathcal{L}_{BCE}(\mathbf{y}, \hat{\mathbf{y}}) = - \sum_{i=1}^{|\mathcal{L}|} \alpha_i y_i \log(\hat{p}_i) + (1 - y_i) \log(1 - \hat{p}_i). \quad (3.52)$$

As mentioned in [Section 3.2](#), binary classification can suffer from severe label imbalance, if there exists a high imbalance between the positive and negative class. In multi-label classification via binary relevance this problem can possibly occur for each of the $|\mathcal{L}|$ binary classification.

The positive weight α_i constitutes a hyperparameter which is used to rebalance the loss of the positive and negative classes. By choosing $\alpha_i > 1$ we disproportionately penalize false positives of label i . An intuitive approach might be to choose each α_i inversely proportional to the label distribution corresponding to label i in the training data. Though, in severe cases of imbalance this leads to very high values of α_i , consequently encouraging predictions with high recall but possibly very poor precision. Additionally, by making the values of α_i dependent on the training data distribution, we implicitly assume the training data distribution to be a good estimate for the population. Given the methods of *Active Learning* and *Resampling* described in [Section 3.2](#), that are employed to combat label sparsity, this premise might knowingly be false to begin with. Another way of dealing with severe class imbalance is the application of threshold tuning as described in [Section 3.2](#), which does not actively influence the model weights.

FOCAL LOSS: The *Focal Loss* loss function was first introduced by Lin et al. in context of computer vision and object detection [[Lin+17](#)]. Additionally to hyperparameter α_i used in the [BCE](#) loss, *Focal Loss* introduces a *focussing* hyperparameter γ . The hyperparameter smoothly applies a downweighting to the easy examples and thus shifts the focus of the training to the misclassified examples.

$$\mathcal{L}_{FL}(\mathbf{y}, \hat{\mathbf{y}}) = - \sum_{i=1}^{|\mathcal{L}|} \alpha_i y_i (1 - \hat{p}_i)^\gamma \log(\hat{p}_i) + (1 - y_i) \hat{p}_i^\gamma \log(1 - \hat{p}_i) \quad (3.53)$$

In [Figure 3.12](#) the loss produced by [BCE](#) loss and *Focal Loss* with focussing parameter $\gamma = 2$ and $\gamma = 4$ is shown for a true label $y = 1$ over predicted probabilities $\hat{p} \in [0, 1]$.

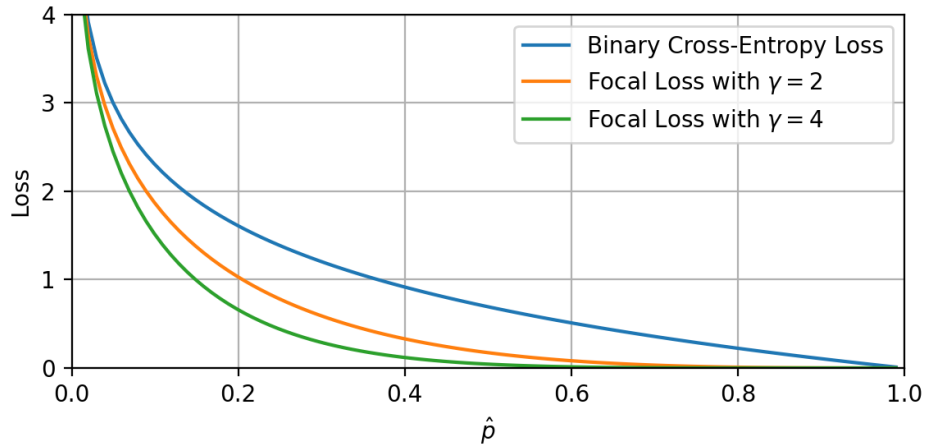


Figure 3.12: *Binary Cross-Entropy Loss* and *Focal Loss* for different predicted probabilities \hat{p} and true label $y = 1$

The curve corresponding to focal loss is heavily dampened in the center region corresponding to uncertain correct predictions or slightly wrong predictions (assuming a decision threshold $t = 0.5$). Thus, the low region of \hat{p} corresponding to misclassified examples has a much a higher relative contribution to the overall loss than in the case of [BCE](#) loss.

In [Section 3.3.4](#) we have seen that LLMs contain large amounts of parameters, which makes (pre-)training and deploying these models very resource- and compute-intensive. For one, models with high parameter counts have correspondingly high memory requirements, which lead to higher costs and might even be infeasible to provide in resource constrained environments like edge devices. Additionally, and most importantly for our work, the number of numerical operations required for a single forward-pass through the model directly corresponds to the number of model parameters. Parallel processing techniques like *Single Instruction Multiple data (SIMD)* on CPUs as well as the highly parallel architecture of GPUs accelerate the processing of vectorized numerical operations found in neural networks. Though, even with highly parallel hardware accelerating neural networks, the inference times of LLMs might range up to multiple seconds depending on the specific model architecture, batch size and maximum sequence length. Various approaches to compress neural networks have emerged, which enable a reduction of model size as well as the acceleration of model inference times. Namely these are *quantization*, *knowledge distillation* and *pruning*. In the following subsections we will focus on the first two methods.

4.1 QUANTIZATION

Quantization in context of neural networks, or digital signal processing in general, is a compression technique used to represent data with reduced numerical precision. Data used in weights, parameters of activation functions and inputs of neural networks is typically represented as floating point data type with a bit width of 32 bits. There exists a wide range of quantization approaches with varying strengths of bit width reduction. In *Mixed Precision Training* the data is represented by a mixture of full- and half-precision floating point values with a bit width of 32 bits and 16 bits, respectively. This enables *Tensor Processing Units (TPUs)* and GPUs to process larger batch sizes as well as larger model architectures with only marginal loss of accuracy. More extreme approaches are binarized [CB16] or ternarized [HS14] neural networks, constraining the data to $x_{bin} \in \{0, 1\}$ or $x_{tern} \in \{-1, 0, 1\}$, respectively. Though, extreme low-bit quantization approaches are often accompanied by significant accuracy loss, especially in context of complex architectures like RNNs and deep neural network architectures [Den+20].

Consequences of quantization and bitwidth reduction are the decrease of required disk and memory space, inference latency and possibly a reduction in model accuracy. A specifically interesting form of quantization is *Integer*

Quantization where floating point values are transformed into a suitable integer representation. A bit width of 8 bits for the result data type has shown to be a good compromise between minimal accuracy loss, significant latency improvement and a memory reduction by a factor of four [Jac+18]. Additionally, modern CPUs provide highly optimized instruction set extensions for vectorized integer operations, namely the *Advanced Vector Extensions SIMD (AVX)* with the *Vector Neural Network Instructions (VNNI)*¹ for the execution of 8 bit vector and matrix calculations. In context of this work, we will focus on 8 bit integer quantization to optimize the CPU inference latency of LLMs.

The selection of an optimal quantization scheme for specific weights and activations is a balancing act between the introduction of minimal computational overhead and an ideal representation of the underlying distribution of values. Generally, integer quantization is the mapping of a real value r to a quantized representation Q . We distinguish between *symmetric* and *asymmetric* as well as *uniform* or *linear* and *non-uniform* quantization. In this work we will focus on *uniform symmetric quantization* which is the most common and the least computationally expensive variant.

4.1.1 Symmetric vs. Asymmetric Quantization

Asymmetric or *affine* quantization of a real value r can be formalized as

$$Q = \left\lfloor \frac{r}{S} \right\rfloor + Z \quad (4.1)$$

with the integer zero point Z and the scaling factor S . The operator $\lfloor \cdot \rfloor$ is rounding input values to the next integer and the scaling factor S is defined as

$$S = \frac{\beta - \alpha}{2^b - 1} \quad (4.2)$$

with $[\alpha, \beta]$ being the real valued input range. Input values outside this range are clipped to the lower or upper boundary with parameter b being the quantization bit width.

Symmetric quantization can be understood as special case of *asymmetric quantization* with integer zero point $Z = 0$. In symmetric quantization the input range is defined as $[-\alpha, \alpha]$ with

$$\alpha = \max(|x|) \quad (4.3)$$

and with x being the set of input values to be quantized ie. weight layers or activation parameters. In Figure 4.1 both symmetric and asymmetric quantization of real valued ranges can be seen.

¹ <https://www.intel.com/content/www/us/en/developer/articles/guide/deep-learning-with-avx512-and-dl-boost.html> last viewed on 26.03.2023

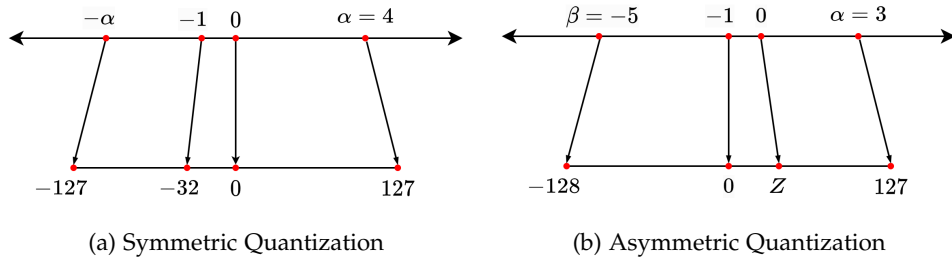


Figure 4.1: Symmetric and Asymmetric Quantization

Even though asymmetric quantization is able to more expressively capture the input distribution, Jacob et al. find that compared to symmetric quantization it poses significant additional operational overhead [Jac+18].

4.1.2 Uniform vs. Non-Uniform Quantization

Another distinction has to be made between linear or uniform quantization and non-uniform quantization. Both quantization schemes are shown in Figure 4.2. In both cases a step function resulting from the quantization, or concretely the application of the integer operator shown in Equation 4.1, can be seen. The distances between the quantized values are equivalent in the uniform case and vary in the non-uniform case. Thus, the scaling factor S is a scalar in the uniform case and a function of r ie. $S(r)$ in the non-uniform case. When dealing with input data that is unevenly spaced, non-uniform quantization is able to better capture important regions.

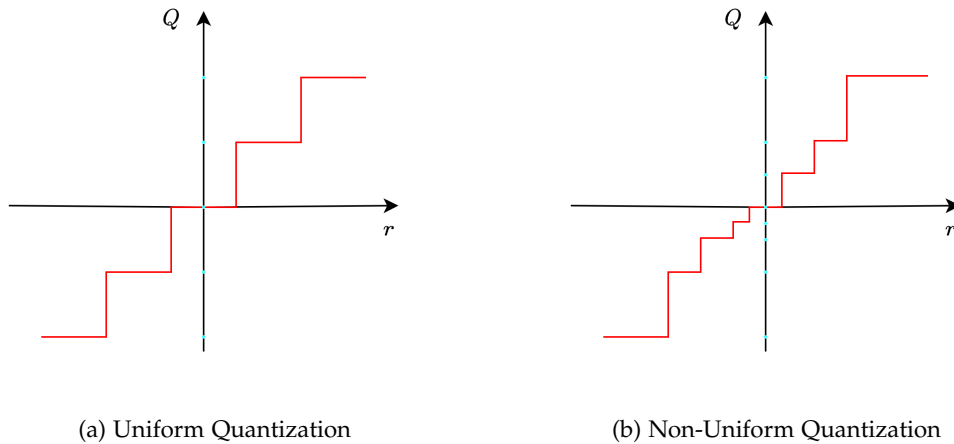


Figure 4.2: Uniform and Non-Uniform Quantization

4.1.3 Post-Training Quantization vs. Quantization-Aware Training

Quantization can be applied at different stages of the model lifecycle. In *Quantization-Aware Training* the quantization parameters constitute additional parameters to be learned during training. *Post-Training Quantization (PTQ)*

on the other hand is, as the name implies, applied after the model is fully trained. *PTQ* methods can generally be separated into static and dynamic methods. Following, we provide a summary for the different methods.

STATIC QUANTIZATION: Static Quantization requires calibration data that is input to the model to compute the quantization parameters for the activations. These parameters are then included as constants in the execution graph of the quantized model and as such are used for every input.

DYNAMIC QUANTIZATION: In *Dynamic Quantization* scale S , zero point Z and the range $[\alpha, \beta]$ for activation parameters is calculated at runtime. Scale and zero point for weights however is calculated during compile time. Dynamic Quant. typically higher inference time but also higher accuracy.

QUANTIZATION AWARE TRAINING The quantization step can be incorporated into the training of the network, which is known as *Quantization Aware Training*. This is done by wrapping layer and activation nodes of the computational graph of a floating-point neural network model with quantization and dequantization nodes and performing simulated or fake quantization during training. Thereby optimal model and quantization parameters are learned with the latter being applied downstream in the actual quantization of the model [Jac+18]. Unfortunately, for large models the computational cost of the *Quantization Aware Training* approach can be prohibitively high, potentially requiring hundreds of epochs of retraining to recover the models accuracy.

4.2 KNOWLEDGE DISTILLATION

Analogous to the previously discussed quantization, *Knowledge Distillation* is an approach to reduce the resource footprint of a trained model through compression. In *Knowledge Distillation* a student model is employed to learn from a trained teacher model. Teacher and student can be part of the same architecture family, which is the case for *DistilBERT*, a reduced version of [BERT](#), but could also be of different model type. The teacher might also be an ensemble of models. The latter equates to the approach we took in this work by distilling the knowledge of binary teachers to a single multi-label model as described in [Section 3.2.2](#). In this section we will specifically look at the distillation procedure for *DistilBERT*, since this is one of the models we employ as final multi-label classification model.

Knowledge Distillation dates back to Bucila et al. [BCNMo6], who showed that it is possible to distill the knowledge contained in an ensemble of models into a single student model. The general knowledge distillation framework is visualized in [Figure 4.3](#).

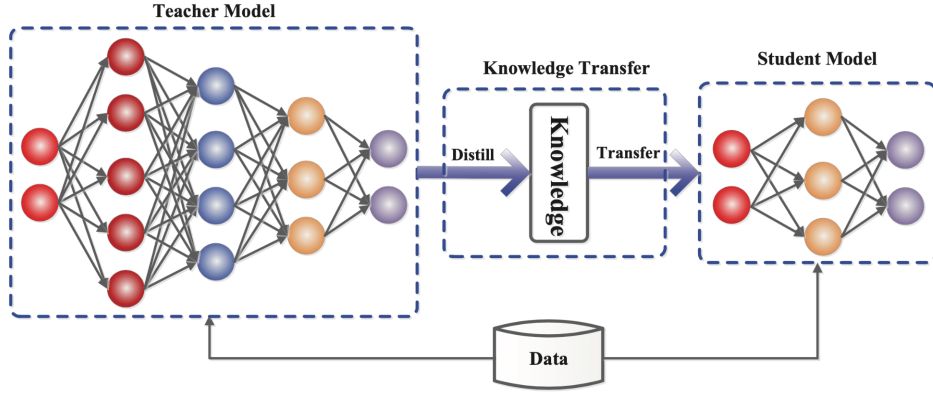


Figure 4.3: Knowledge Distillation [Gou+21]

To train a student model with a given teacher and ground truth data Hinton et al. [HVD15] propose a combined loss function consisting of a weighted average of two losses. The first loss function is the cross-entropy between the teachers and students probability outputs that are softened by a temperature parameter T . Thus, a softened probability output for class $i \in c$ is calculated by applying the modified softmax function $\sigma(\mathbf{z})$ to the logit outputs \mathbf{z} as

$$\begin{aligned} q_i &= \sigma_i(\mathbf{z}/T) \\ &= \frac{\exp(z_i/T)}{\sum_{j=1}^c \exp(z_j/T)}. \end{aligned} \quad (4.4)$$

The first loss function minimizes the Kullback-Leibler divergence between the teacher and student and hence is abbreviated as \mathcal{L}_{KL} . It is defined as such

$$\mathcal{L}_{KL}(\mathbf{z}_t, \mathbf{z}_s) = -T^2 \sum_{j=1}^c \sigma_j(\mathbf{z}_t/T) \log \sigma_j(\mathbf{z}_s/T) \quad (4.5)$$

with \mathbf{z}_t being the logit outputs of the teacher and \mathbf{z}_s being the logit outputs of the student. The second loss is the cross-entropy loss, \mathcal{L}_{CE} , of the students probability outputs and the true labels

$$\mathcal{L}_{CE}(\mathbf{y}, \mathbf{z}_s) = - \sum_{j=1}^c y_j \log \sigma_j(\mathbf{z}_s). \quad (4.6)$$

The combined loss is the weighted sum of both losses

$$\mathcal{L} = \alpha \mathcal{L}_{KL} + \beta \mathcal{L}_{CE} \quad (4.7)$$

with the weighting factors α and β .

DistilBERT

Sanh et al. show that it is possible to train a student model based on BERT [Dev+18] with a size reduction of 40%, but at the same time a performance

retention of 97% of the original BERT model. The student model called *DistilBERT* is generally of the same architecture as the BERT teacher model. The student is initialized by copying every second layer of the teacher model into the student and as such reducing the number of layers from 12 to 6.

The loss function utilized is a combined loss as described in Section 4.2. In addition to the KL-loss between the tempered softmax output of the teacher and student, \mathcal{L}_{KL} , and the CE-loss of the student outputs and ground truth labels, \mathcal{L}_{CE} , a cosine embedding loss, \mathcal{L}_{cos} is integrated into the total loss function. This loss function is used to regularize the directions of the hidden state vectors of student and teacher. As such the total loss is defined as

$$\mathcal{L} = \alpha\mathcal{L}_{KL} + \beta\mathcal{L}_{CE} + \gamma\mathcal{L}_{cos}. \quad (4.8)$$

In contrast to the pre-training of BERT, Sanh et al. only apply the *masked-language-modeling* objective without the *next-sentence prediction* objective. They find that removing the NSP loss from the training procedure leads to equal or even better performance on downstream tasks.

CONCEPT AND EXPERIMENTAL SETUP

In this chapter we present an end-to-end workflow for the creation of a scalable multi-label classification system based on CPU latency-optimized LLMs and trained on PL multi-label data. A high-level overview of the proposed workflow is shown in Figure 5.1.

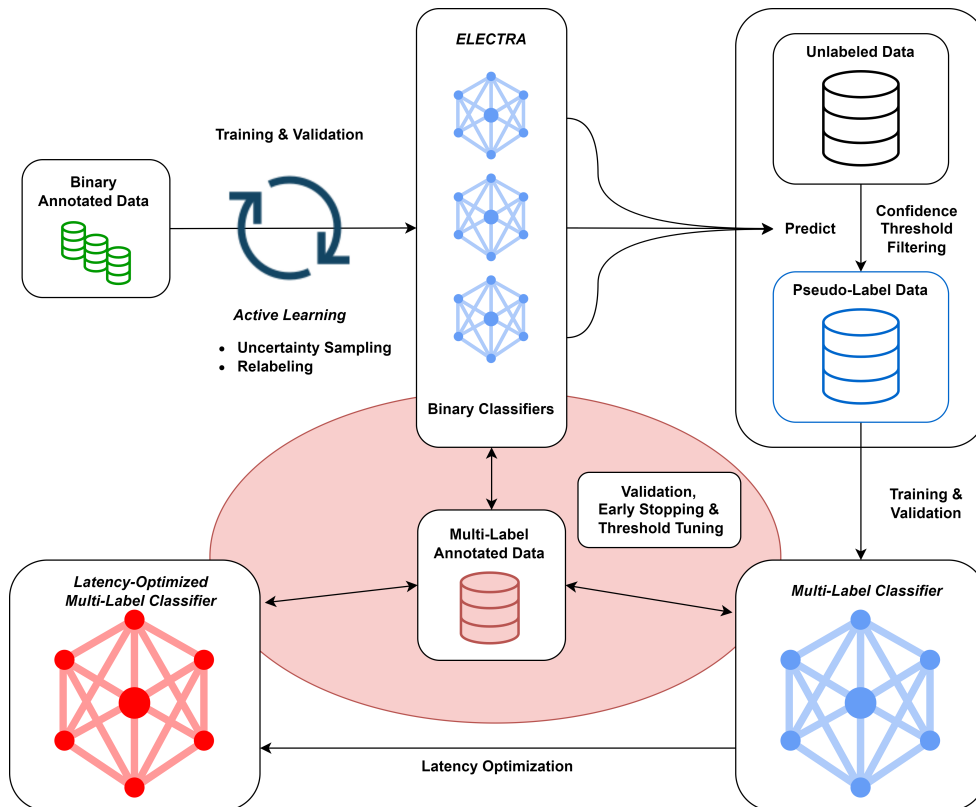


Figure 5.1: End-to-End Pipeline

In the following sections, we describe the end-to-end process in detail, beginning with the datasets employed in our experiments as well as the corresponding binary annotation procedure. We look at the batchwise *Active Learning* component to optimize the training data corresponding to binary classification models with unsatisfying performance. Following, we present the curation of *Pseudo-Label* datasets by employing optimized binary classifiers and *Confidence Threshold Filtering*. We also give an overview of the baseline models utilized for binary and multi-label classification, whose performance measures serve as reference benchmark for the transformer-based models. Further, we look at the early-stopping criterion in the training of binary and multi-label transformers. We conclude this chapter with

an overview of the methods applied to optimize the inference latency of the trained large language models.

5.1 DATA

The data used in our experiments is a large collection of german textual customer feedbacks in context of regional and long-distance traffic of *Deutsche Bahn*. Customers submit feedbacks through different channels, which are a mobile application, *DB Navigator*, the *ICE portal* accessible from within long-distance trains and via QR codes placed on the seats of long-distance trains encoding the URL to a feedback submission form. The feedback form provides two text fields with one of the fields purposed for positive feedback and the other for negative feedback. The textual feedbacks used in our experiments are already preprocessed with regards to privacy and compliance requirements, meaning that potentially tracable personal information is removed from the feedbacks. The anonymized feedbacks are persisted in a data warehouse and amount to multiple millions of unlabeled samples. To provide a better perspective on the data at hand, we present a selection of synthetic exemplary samples with potential categories in brackets:

Synthetic Feedback Examples

- *Delays!* - [Punctuality]
- *The train stopped unexpectedly and no one knew what's happening. Fortunately, the train attendant was very nice and provided us with information. Kind regards, XXX XXX* - [Train Service]
- *WLAN is working, seats are comfortable* - [WLAN / Internet, Seating Comfort].
- *The train was late and my connection is canceled. Now, I'll be over an hour late. I want my money back!* - [Punctuality, Train Cancellation, Passenger Rights]

5.1.1 Binary-labeled training data

In addition to the data warehouse containing large amounts of unlabeled data, we also have access to expert-labeled data for around 100 categories, which are split into a shallow hierarchical structure with a main category being parent to potentially multiple descendants. The existing annotations are binary as they are typically created in context of projects in which a specific category or topic has to be investigated. As an example, one of these projects might be the investigation of recent trends in customer sentiment regarding WLAN availability. The annotations are performed with an internal application that enables annotators to query the data warehouse with key-

word searches and through weak labeling with decision trees. The selection of weakly labeled examples is then re-annotated by the expert annotator.

| Category Name | Mean | Std. Dev. | Min | Max | Documents with > 512 Tokens |
|---------------------|-------|-----------|-----|-----|--------------------------------|
| Face Mask General | 38.29 | 34.69 | 3 | 222 | 0 |
| Loudness | 30.39 | 36.73 | 3 | 606 | 1 |
| Luggage General | 34.57 | 40.79 | 4 | 606 | 1 |
| Passenger Rights | 46.34 | 61.41 | 5 | 789 | 2 |
| Punctuality | 26.94 | 29.04 | 3 | 198 | 0 |
| Seat Availability | 17.77 | 18.23 | 3 | 297 | 0 |
| Seating Comfort | 30.34 | 42.41 | 3 | 606 | 1 |
| Temperature General | 27.00 | 34.78 | 3 | 606 | 1 |
| Train Cancellation | 28.30 | 25.83 | 3 | 288 | 0 |
| Train Service | 20.76 | 23.98 | 3 | 361 | 0 |
| WLAN / Internet | 23.88 | 21.75 | 3 | 276 | 0 |

Table 5.1: Token count statistics per category

We choose a subset of ten categories with the selection criteria being the amount of available annotated data and topic diversity. In addition to the ten existing categories, we introduce a new category, *Passenger Rights*, and curate a corresponding training dataset by iteratively extending the dataset in an *Active Learning* loop as explained in [Section 3.2.1](#), until we find the performance improvement through dataset extension to be plateauing. The evaluation results of this process are presented in [Chapter 6](#). The selected categories and descriptive statistics concerned with document length, specifically the amount of (sub-)word tokens contained in the documents after tokenizing them with a *WordPiece* [Wu+16] tokenizer, can be seen in [Table 5.1](#). We include a column showing the number of documents with a token count greater than 512. This is an important threshold, since the *BERT*-based models, *ELECTRA* and *DistilBERT*, we introduced in [Section 3.3](#) and [Section 4.2](#), respectively, are pre-trained on a maximum sequence length of 512 (sub-)word tokens. Processing longer documents would require us to either discard parts of the documents or to include document splitting and a subsequent combination of the classification results on the sub-documents. Yet, only a minimal amount of samples exceeds this threshold, which is why we opt to truncate these samples.

| Category Name | n_{sample} | n_{pos} | $frac_{pos}$ |
|---------------------|--------------|-----------|--------------|
| Face Mask General | 1036 | 182 | 0.18 |
| Loudness | 10765 | 1366 | 0.13 |
| Luggage General | 12193 | 2000 | 0.16 |
| Passenger Rights | 1500 | 768 | 0.51 |
| Punctuality | 16671 | 4771 | 0.29 |
| Seat Availability | 14265 | 3582 | 0.25 |
| Seating Comfort | 10694 | 1193 | 0.11 |
| Temperature General | 11199 | 1542 | 0.14 |
| Train Cancellation | 5631 | 1728 | 0.31 |
| Train Service | 12590 | 3243 | 0.26 |
| WLAN / Internet | 10865 | 1628 | 0.15 |

Table 5.2: Sample and label distribution within the binary training data of eleven categories

As described in [Section 3.2.3](#), there are multiple methods to combat performance degradation resulting from severe label imbalance. We employ re-sampling, threshold tuning and loss reweighting to specifically improve the *precision* of the resulting classifiers.

Sample intersection

The union of the eleven binary-annotated datasets consists of $n = 31620$ unique samples with only one third of the samples appearing in multiple datasets. In [Table 5.3](#) we can see that there isn't a single sample that was annotated in context of all eleven categories. A third of the samples appear in more than one binary dataset with the majority of multi-labeled samples appearing in exactly seven categories. This makes the creation of multi-class or multi-label models challenging, since these models require fully annotated training data.

| Number of categories | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|--------------------------|-------|-----|----|-----|----|-----|------|-----|---|----|----|
| Number of unique samples | 20966 | 650 | 18 | 180 | 39 | 458 | 9148 | 160 | 1 | 0 | 0 |

Table 5.3: Size of intersections between binary-annotated datasets

To circumvent the challenge of partial labels, we create fully labeled pseudo-label data with trained binary classifiers and filter the data with an uncertainty threshold measure.

5.1.2 Multi-label data

As explained in the previous section, the intersection between the binary-annotated datasets generally decreases with an increase of the number of categories with none of the samples appearing in all of the eleven datasets. To distill the knowledge of the binary classifiers into a single multi-label model, we employ *pseudo-labeling* to generate a fully labeled dataset used as training data for the multi-label model. Given time and resource constraints, we use a subset of the available unlabeled data in the data warehouse consisting of 200,000 samples.

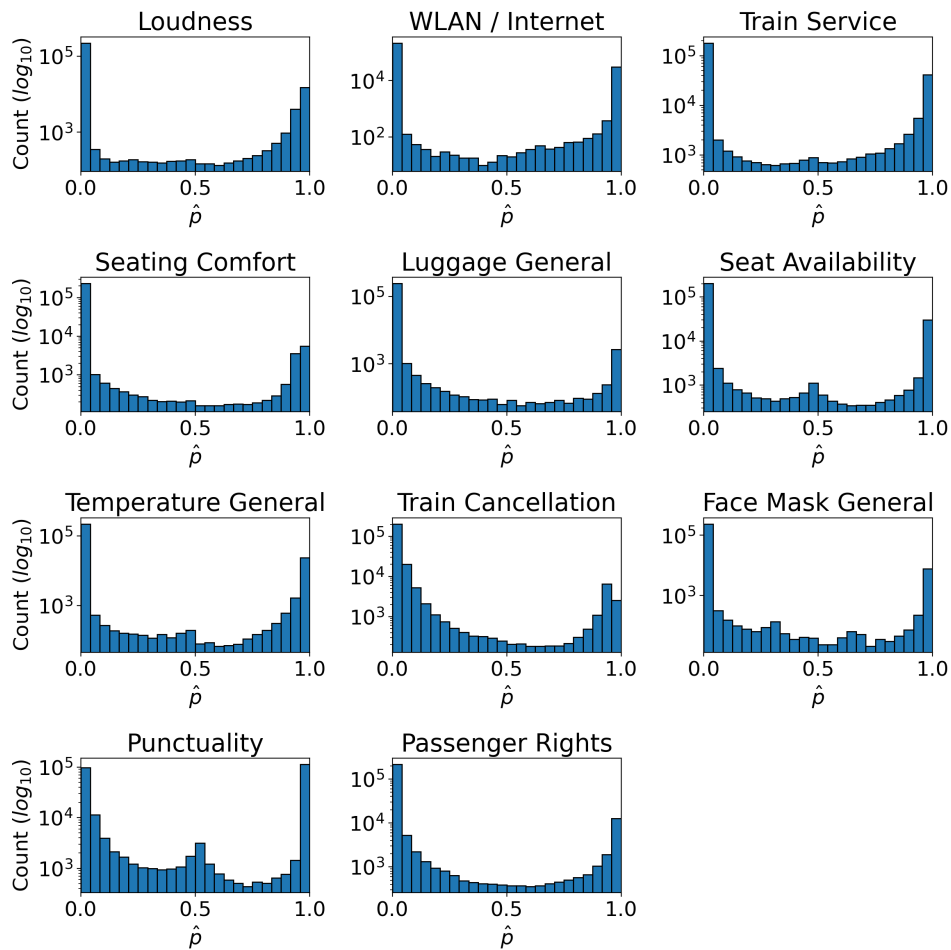


Figure 5.3: Distribution of predicted probabilities on unlabeled data

Each trained and optimized binary classifier is used to perform a binary prediction on each unlabeled sample. The detailed training, evaluation and prediction procedure will be explained in the following [Section 5.2](#). The distribution of the predicted probabilities per category is shown in [Figure 5.3](#) with the y-axes representing the number of samples on a logarithmic scale. Each of the barplots displays a bimodal distribution with the modes around zero and one, respectively. Though, the histogram corresponding to *Punctuality*

shows many uncertain samples with predicted probability around 0.5. Further, the probability mass around zero is higher than around one, which is to be expected given the imbalanced label distribution of the expert-annotated data shown in Table 5.2. We use the *Scaled Confidence Score*, as explained in Section 3.2.2, to filter for the most certain predictions using a probability threshold α .

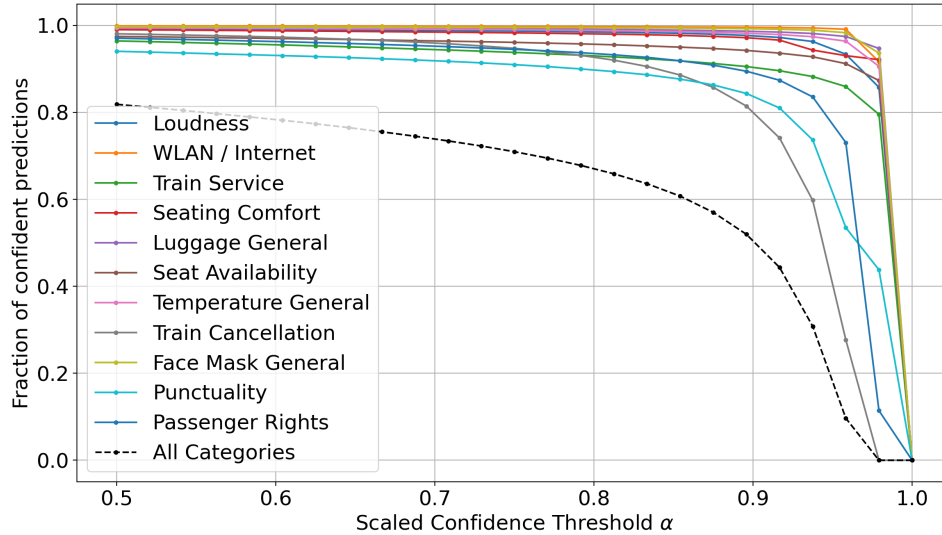


Figure 5.4: Fraction of samples with *Scaled Confidence Score* above threshold α

We calculate the *Scaled Confidence Score* for each sample and each category and set the confidence threshold to $t = 0.85$. This means that only samples with corresponding *Scaled Confidence Score* greater than t for every category will be included in the multi-label training data.

| Category Name | n_{pos} | $frac_{pos}$ |
|---------------------|-----------|--------------|
| Face Mask General | 1865 | 0.05 |
| Loudness | 3367 | 0.09 |
| Luggage General | 778 | 0.02 |
| Passenger Rights | 2328 | 0.06 |
| Punctuality | 17880 | 0.48 |
| Seat Availability | 5010 | 0.13 |
| Seating Comfort | 2743 | 0.07 |
| Temperature General | 4539 | 0.12 |
| Train Cancellation | 2270 | 0.06 |
| Train Service | 7864 | 0.21 |
| WLAN / Internet | 5192 | 0.14 |

Table 5.4: Label distribution of the confidence-filtered pseudo-label multi-label data consisting of 37,621 samples

We choose this specific threshold as a compromise between predictive confidence and abundance of positively labeled data. Since we already apply relatively strict constraints by requiring each individual label prediction to exceed the specified threshold, choosing the threshold more conservatively would lead to very low positively labeled examples for the more sparse categories like *Luggage General*. This could potentially be alleviated by using larger amounts of unlabeled data, which, however, was out of scope for this work. The final multi-label dataset consists of 37,621 samples with the label distribution shown in Table 5.4. Compared to the label distributions of the binary expert-annotated data, as shown in Table 5.2, we find a decreased fraction of positives for most of the categories. This is a result of the sampling procedure applied for the curation of the binary datasets as explained in Section 5.1.1. In Figure 5.5 the pairwise pearson correlations between the label columns y_i of the pseudo-label data are shown.

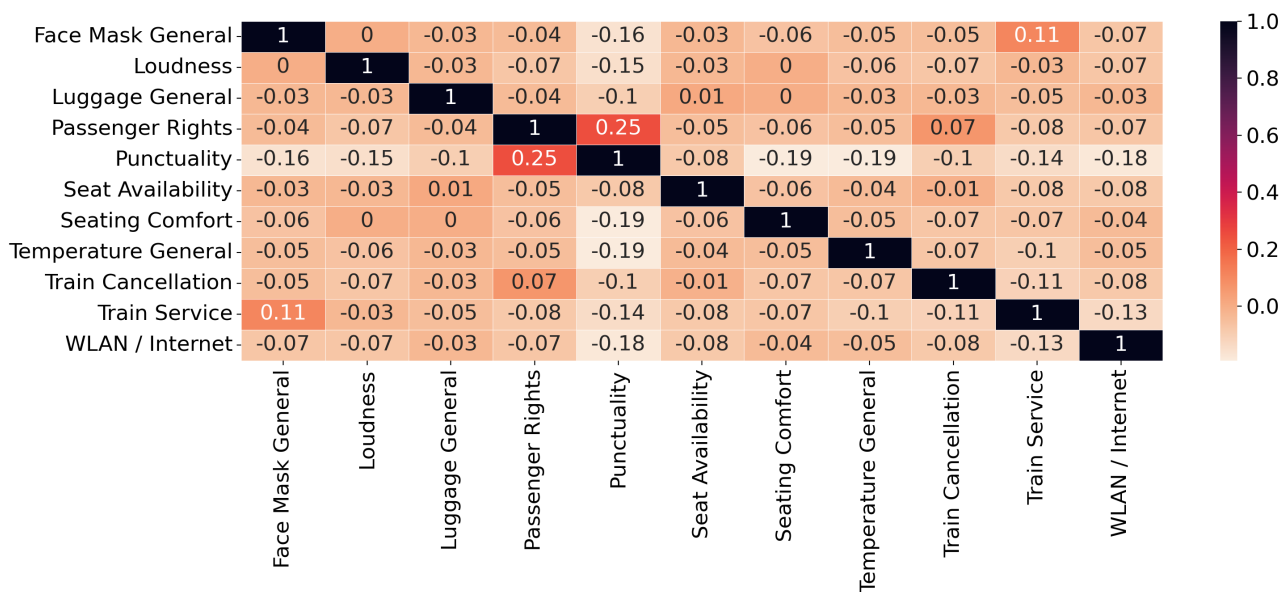


Figure 5.5: Label correlation $P(Y)$ found in the pseudo-label multi-label data

Most of the label columns show a minor negative correlation. This is because most of the feedbacks are relatively short with a majority of feedbacks being associated with a single category. Exceptions to this are the pairwise correlation between *Train Service* and *Face Mask General*. This makes sense, since feedbacks associated with *Face Mask General* often contain complaints about a too limited enforcement of the mask mandate regarding other passengers or a too rigid enforcement of the mask mandate regarding themselves. The other two positive pairwise correlations of *Passenger Rights* with *Punctuality* and *Train Cancellation*, respectively, are also reasonable, since *Passenger Rights* is comprised of feedbacks mentioning delays over one hour (*Punctuality*) as well as feedbacks mentioning cancelled trains, leading to additional costs (*Train Cancellation*), among others.

Dataset Splits

A standard practice in supervised learning is the utilization of dataset splits, where one data partition is used for training and the remaining partition is used for model evaluation. We utilize *5-fold stratified cross-validation*, where the dataset is partitioned into five partitions of equal size and equal label distribution. This is done via the *Iterative Stratification* algorithm presented by Sechidis et al. [STV11]. The algorithm first calculates the desired number of samples per fold and label and continues by iteratively filling the partitions with samples associated with the rarest label.

In addition to the cross-validation split $\mathcal{D}_{val_{cv}}$, we manually annotate an additional multi-label *validation dataset* consisting of 800 samples, $\mathcal{D}_{val_{ood}}$. One half consists of samples with low confidence scores and the other half is a random sample. This validation dataset plays an important role in the early stopping procedure we include in the training, which will be explained in the next section. Finally, we also use a randomly sampled and manually annotated *test dataset*, \mathcal{D}_{test} , which is used to benchmark the performance of the different models after their training is completed.

Utilized datasets

| | |
|---------------------------|---|
| $\mathcal{D}_{val_{cv}}$ | Cross validation fold used to evaluate the training progress. |
| $\mathcal{D}_{val_{ood}}$ | Additional validation dataset with difficult examples used for training evaluation and early stopping. |
| \mathcal{D}_{test} | Holdout test dataset used to ultimately evaluate the predictive performance of the different model architectures. |

Bias

As explained in [Section 3.2.2](#), we employ multiple measures to reduce the amount of *label noise* and *bias* affecting the intermediary *pseudo-label* data. We use high-capacity *transformer* models as label-generator models and an early stopping procedure to estimate the generalization of the model in training and consequently avoid the training of overfitted models.

The filtering of uncertain pseudo-label data reduces the amount of label noise in the data. However, a very conservative threshold might lead to a highly biased dataset, keeping only very simple examples. Additionally, the sampling and annotation procedure used to curate the binary training data by employing *distant supervision* and re-annotation comes with inherent bias. Thus, we employ multiple evaluation datasets with different label distributions to detect and combat bias or overfitting-dependent performance degradation.

5.2 TRAINING, EVALUATION AND PREDICTION

In this section we lay out the training and evaluation procedure performed in context of binary and multi-label models. For the binary classifiers we use expert-annotated data as training, evaluation and test data. We extend the dataset corresponding to category *Train Cancellation* with an *Active Learning* loop, as described in the previous section, and, following the same procedure, construct the binary dataset for category *Passenger Rights*. For the multi-label classifiers we use *Pseudo-Label* data as training and validation data. Additionally, we employ expert-annotated validation and test datasets, with the latter serving as benchmark dataset across all models.

Regarding text preprocessing, most of the work has to be performed in context of the baseline models, which will be explained in [Section 5.2.1](#). The LLM classifiers do not require extensive text preprocessing except the removal of *HTML* substrings and special characters resulting from different text encodings. We also perform lower casing on the input text for the fine-tuning of models pre-trained on lower case german text.

The training, evaluation and tracking of the binary and multi-label neural network models, ie. *Transformer* and *LSTM* models, follows the steps summarized in [Algorithm 1](#). The general procedure consists of batchwise feed-forward passes of the input data through the neural network, gradient calculation via backpropagation and subsequent model weight updates with an optimizer such as *Adam* [KB14]. We evaluate the model performance either at the end of each training epoch, ie. after the model has processed the full training dataset \mathcal{D}_{train} , or after a fixed number of batches depending on the size of the training dataset. We closely monitor the training by propagating the evaluation results consisting of the *F1-score*, *recall*, *precision* and the current *loss*, among others, to a *MIFlow*¹ tracking server. This server provides a user interface component with graphical and tabular representations of tracked experiment results. Real-time experiment monitoring makes it possible to manually intervene and restart the training in case of unsatisfactory results or abnormal behaviour. Additionally, it provides a structured history of previous experiment parameters and evaluation results. For the training of the neural network models we use a *NVIDIA GeForce RTX 2080 Super* graphics processing unit with 8 GB virtual random access memory. This enables us to train the models with batches of size 8 for the larger *ELECTRA* and a batch size of 16 for the smaller *DistilBERT*. To simulate training with larger batch sizes with smoother weight updates, we also include gradient accumulation in our training.

¹ <https://mlflow.org/> (last visited on 26.03.2023)

Algorithm 1 Training-, Evaluation- and Tracking-Loop

Require: Model $model$
Require: Optimizer opt
Require: $EarlyStopper$
Require: Loss function \mathcal{L}
Require: Thresholds \mathbf{t}
Require: Datasets $\mathcal{D}_{train}, \mathcal{D}_{val_{cv}}, \mathcal{D}_{val_{ood}}$
Ensure: $F1_{best} \leftarrow 0$

- 1: **for each** $epoch \in epochs$ **do**
- 2: **for each** batch $(x, y) \in \mathcal{D}_{train}$ **do**
- 3: $\hat{y} \leftarrow model(x)$ ▷ Forward-Pass
- 4: $loss \leftarrow \mathcal{L}(y, \hat{y})$
- 5: $gradients \leftarrow backprop(loss, model)$
- 6: $model \leftarrow optimize(model, gradients, opt)$
- 7: **if** $shouldEvaluate()$ **then**
- 8: $(F1_{cv}, _) \leftarrow eval(model, \mathcal{D}_{val_{cv}}, \mathbf{t})$
- 9: $(F1_{ood}, \mathbf{t}_{new}) \leftarrow eval(model, \mathcal{D}_{val_{ood}}, \mathbf{t})$ ▷ Threshold tuning
- 10: $F1_{avg} \leftarrow (F1_{cv} + F1_{ood})/2$
- 11: **if** $F1_{avg} > F1_{best}$ **then**
- 12: $F1_{best} \leftarrow F1_{avg}$
- 13: $\mathbf{t} \leftarrow \mathbf{t}_{new}$
- 14: $checkpoint(model, path)$
- 15: $EarlyStopper.reset()$
- 16: **else**
- 17: $EarlyStopper.decreasePatience()$
- 18: **end if**
- 19: Propagate evaluation results to tracking service
- 20: **end if**
- 21: **if** $EarlyStopper.shouldStop()$ **then**
- 22: $break$
- 23: **end if**
- 24: **end for**
- 25: **end for**

The presented training loop is an ordinary neural network training loop with the addition of F1-score-based threshold tuning and early stopping.

Early Stopping

We utilize two evaluation datasets, a cross validation fold, $\mathcal{D}_{val_{cv}}$, and an *out-of-distribution* dataset, $\mathcal{D}_{val_{ood}}$. Since the training datasets are constructed by combining keyword searches and distant supervision to filter unlabeled data for potential positive samples, the datasets carry an inherent bias. As such, the cross validation data carries the same sampling bias and is often insufficient for the evaluation of the models performance. We combat this problem by including an additional validation dataset consisting of two partitions, an *i.i.d.* sampled partition and a partition of difficult examples. We call this

the out-of-distribution dataset, $\mathcal{D}_{val_{ood}}$, and estimate the models generalization through constant monitoring. We employ an early stopping procedure based on the average evaluation F1-score of the cross-validation and the out-of-distribution validation datasets. The *EarlyStopper*, as shown in [Algorithm 1](#), is a simple counter that is incremented if the average F1 validation score does not surpass a previous best. If the counter reaches a predefined maximum patience threshold, the training concludes. In [Figure 5.6](#) the early stopping procedure with a patience of 4 is shown.

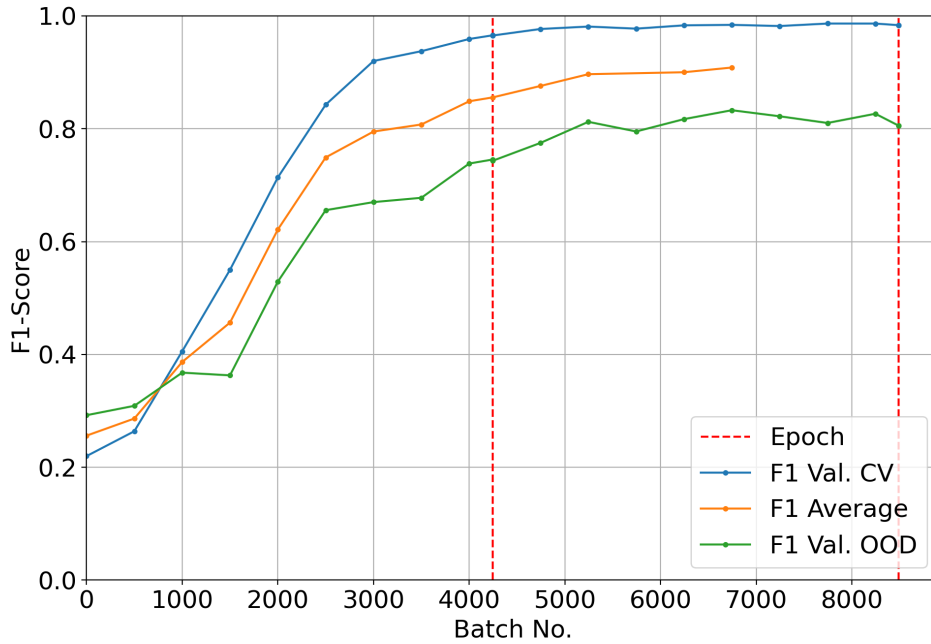


Figure 5.6: Early stopping induced by stagnating averaged F1-score improvement

Each dot on the *average validation* curve (orange) signals an increase in F1-score and the creation of a checkpoint. Following the last orange dot four evaluations without an increase in the average F1 occur until the patience threshold of the *EarlyStopper* is exceeded and the training concludes.

5.2.1 Baseline Models

To provide a performance reference for the binary and multi-label LLM classification models, we also train support vector machine (SVM) [CV95], XGBoost and Long Short-Term Memory (LSTM) models whose evaluation results on the shared test dataset serve as baseline for the following experiments.

svm: To provide a baseline evaluation with a relatively simple model architecture, we employ a SVM classifier with a radial basis function kernel and L2 regularization. We use a count-based text representation to train the SVM classifier.

XGBOOST: After performing a *bag-of-words* preprocessing with count-based methods or through TFIDF-vectorization, as described in [Section 3.3.1](#), we utilize *XGBoost* [CG16] classifiers for binary and multi-label prediction. *XGBoost* is a tree-based ensemble machine learning method that sequentially optimizes a tree-structure of weak learners. It is a highly adaptive algorithm that tends to perform well on a wide range of classification and regression tasks. We perform extensive hyperparameter tuning by employing a Tree-structured Parzen Estimator [Ber+11] algorithm implemented in the hyperparameter tuning library *Optuna*². Instead of exhaustively searching through a fully specified grid of hyperparameter values, the Tree-structured Parzen Estimator algorithm tries to maximize the *Expected Improvement* criterion with a statistical learning procedure.

BIDIRECTIONAL LSTM: We also utilize bidirectional LSTM models [HS97], which in contrast to the bag-of-words approach used with *XGBoost* are able to utilize local context information within sequential inputs. For the training of binary and multi-label LSTMs we use a concatenation of pre-trained word embeddings consisting of 300-dimensional german *fastText* and *GloVe* word embeddings [PSM14] trained on a corpus consisting of documents related to *Deutsche Bahn*. The *fastText* embedding procedure is an extension of the *skip-gram* embedding procedure utilizing *n-grams* as described in [Section 3.3.1](#).

5.2.2 Large Language Models

We select the LLM model architectures *ELECTRA* and *DistilBERT* for the classification experiments. Creating the LLM multi-label classification model consists of two major building blocks. We first train binary classification models for each of the individual categories. Here we are not concerned with latency constraints and as such utilize the model architecture yielding the best predictive performance, which we find to be *ELECTRA*. We partition each binary dataset into five stratified folds and train multiple models using different random seeds per split. To create pseudo-labels on an unlabeled corpus consisting of 200,000 samples, we create predicted probabilities with the best model per split. We then average the final probabilities as well as the decision thresholds of the five models. Finally, we create binary predictions by applying the average decision threshold to the averaged predicted probabilities.

5.3 INFERENCE LATENCY OPTIMIZATION

To achieve the overarching goal of enabling real-time multi-label classification with high predictive performance, we first distill the knowledge of multiple binary LLM classifiers into a single multi-label model as described

² <https://optuna.org/> (last visited on 26.03.2023)

in [Section 3.2.2](#). We compare the model architecture selection consisting of *ELECTRA* and its compressed and pruned descendant *DistilBERT*. The execution environment described in [Table 5.5](#) is used to measure the inference latencies of the raw and optimized models. All experiments are performed on CPU hardware.

| Hardware Component | Specification |
|----------------------|---|
| CPU | AMD Ryzen 7 PRO 4750U, 8 cores @ 1.70 GHz |
| RAM | 32GB @ 3.2GHz |
| OS | Windows 10 - 64-Bit |
| Python version | 3.9 |
| PyTorch version | 1.12.1 |
| ONNX Runtime Version | 1.13.1 |

Table 5.5: Execution environment specifications

For latency optimization, the fine-tuned *PyTorch* models are first converted into the framework-agnostic *Open Neural Network Exchange (ONNX)*-format. We utilize the *ONNX Runtime (ORT)*³ and its corresponding CPU execution provider to perform inference on the *ONNX*-converted models. Finally we compress the fine-tuned models into an *dynamically quantized* 8-bit format as explained in [Section 4.1](#).

³ <https://onnxruntime.ai/> (last visited on 26.03.2023)

EXPERIMENTAL RESULTS

In this chapter we present our experimental results. We separate this chapter into two sections. The first part, [Section 6.1](#), deals with the predictive performance of the binary and multi-label classifiers. In [Section 6.1.1](#) we first take a thorough look at the results of the binary classifier for category **Passenger Rights**. This serves as representation of the general procedure and evaluation for all individual binary classifiers, but for brevity, we omit looking at the other binary classifiers in such detail. The binary [LLM](#) classifiers are used to generate the pseudo-label training data for the multi-label models.

Following in [Section 6.1.2](#), we look at the multi-label model results and present the baseline performance results achieved with a multi-label [SVMs](#), hyperparameter-tuned [XGBoost](#) models and bidirectional [LSTMs](#). We then discuss the results of the multi-label [LLM](#) classifiers and investigate changes in performance in comparison to the results of the binary classifiers. With [Section 6.1.3](#) we conclude the first section by presenting the results of a scalability experiment, in which we apply the proposed workflow to a selection of 82 categories.

The second section, [Section 6.2](#), is dedicated to the results of the latency optimization experiments. We measure the inference times of original [PyTorch](#) implementations, converted and optimized [ONNX](#)-representations executed with [ONNX Runtime](#) and 8-bit quantized [ONNX](#)-representations of [LLMs](#) and [LSTM](#) models.

6.1 PREDICTIVE PERFORMANCE

In this section we present the predictive results of the binary and multi-label classifiers. We first look at the evaluation results of the binary classifiers by example of the classifier for category *Passenger Rights*. We highlight the iterative improvement of its predictive performance through the application of a pool-based *Active Learning* loop. We then look at the predictive performance of [SVM](#), hyperparameter-tuned [XGBoost](#) and bidirectional [LSTM](#) multi-label baseline models. Following, we display and compare the performance of the ensemble of binary [LLMs](#), baseline models and multi-label [LLMs](#) in floating point and 8-bit quantized form. We conclude this section with a meta analysis, in which we investigate the scalability of our workflow to train multi-label models with intermediary pseudo-label data by applying the workflow to a total of 82 categories.

6.1.1 Binary classifiers by example of Passenger Rights

Here we evaluate the predictive performance of the binary *Passenger Rights* classifier. The presentation of this specific classifier serves as proxy for the workflow applied to the eleven binary classifiers in a similar fashion. The results of the full set of binary classifiers are discussed in the subsequent multi-label subsection, in which we compare the predictive performance of the binary and multi-label models. We extend the selection of ten existing categories with a new category called *Passenger Rights*. This category is used for customer feedbacks that describe a situation in which the feedback author might be eligible for compensation. In the real-time classification context, the feedback author should receive information and a link to a passenger rights customer form as response to the submitted feedback. To construct the training dataset, we first select weakly labeled positive examples with a word-based search which we then manually re-annotate. We use multiple case insensitive *regular expressions* to filter the unlabeled data for potentially positive examples. A subset of these regular expressions is shown in [Table 6.1](#).

| Description | Regular Expression |
|---|--|
| Customer requires voucher | <code>/.*(verlange antrag möchte gibt will)*.*gutschein.*</code> |
| Customer requires passenger rights form | <code>/.*(gastrecht antrag formular).*/</code> |
| Customer requires refund | <code>/.*(geld kosten betrag preis)\s+zurück.*</code> |

Table 6.1: Exemplary Regular Expressions filtering for feedbacks relevant to category *Passenger Rights*

We also randomly sample the unlabeled data to extend the training data by a partition with a distribution close to the real population. This partition contains mostly negative samples. Additionally, we manually annotate the validation dataset $\mathcal{D}_{val_{ood}}$ and the test dataset \mathcal{D}_{test} with a new column corresponding to the *Passenger Rights* category. In [Table 6.2](#) the improvement of the F1-score on the validation dataset $\mathcal{D}_{val_{ood}}$ corresponding to the addition of *informative* training data is shown. The new training examples are selected satisfying the following criteria:

- The *Scaled Confidence Score* corresponding to the samples is low (*Uncertainty Sampling*).
- The samples contain content with no or very low representation in the current training data.
- Adversarial examples the model currently falsely predicts, leading to false positives and thus low *precision*.

| Number of samples | Number of positive samples | Validation F1-Score on \mathcal{D}_{valood} |
|-------------------|----------------------------|---|
| 1000 | 403 | 0.717 |
| 1151 | 515 | 0.742 |
| 1250 | 622 | 0.772 |
| 1502 | 750 | 0.792 |

Table 6.2: Binary classifier improvement by dataset curation through batchwise *Active Learning*

We find that specifically the addition of adversarial examples to decrease the false positive rate of the model lead to high performance gains. The category *Passenger Rights* is one of the more demanding categories in terms of language and content. The association of samples with this category is highly context-dependent with many edge-cases resulting from the following exemplary criteria:

- Eligibility for compensation depends on the amount of delay.
- Customers demanding a refund might refer to a wrongfully performed ticket purchase, gastronomical services or parking costs not necessarily related to railway travel.
- Mentioned delays or costs might be unrelated to *Deutsche Bahn*.

The following synthetic examples are meant to highlight these challenges:

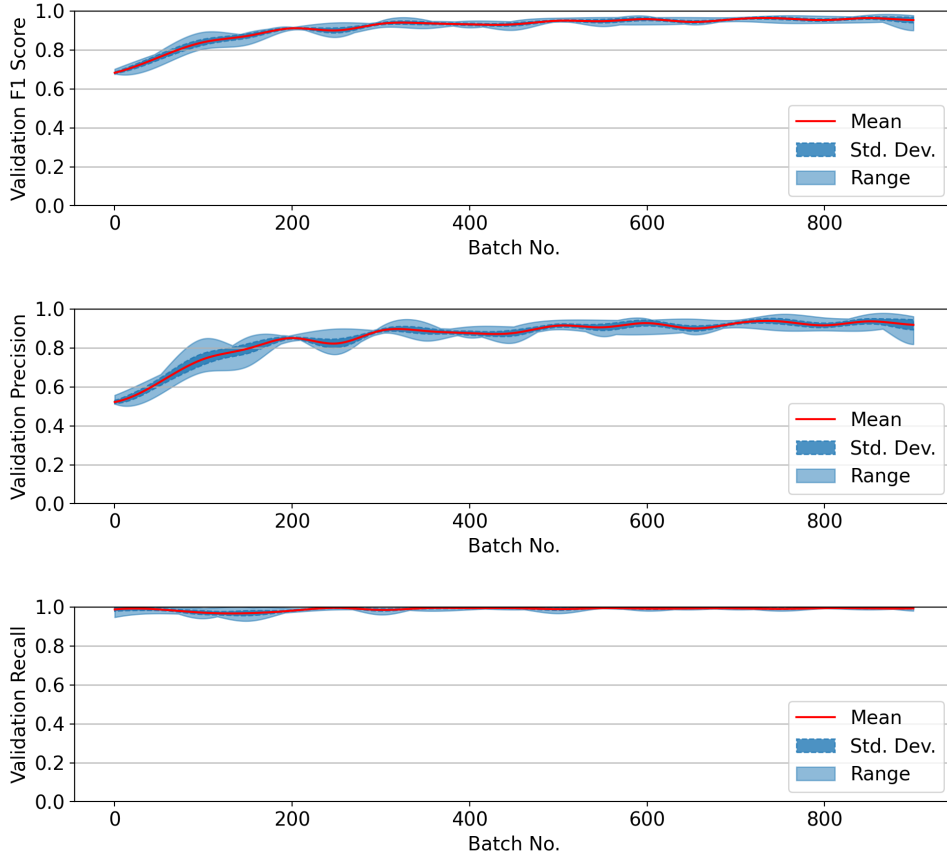
Not associated with Passenger Rights

- *I arrived almost an hour late.*
- *There has been a duplicate booking in the app. I want a refund!*
- *The parking lot was full, so I had to park in an expensive parking garage.*

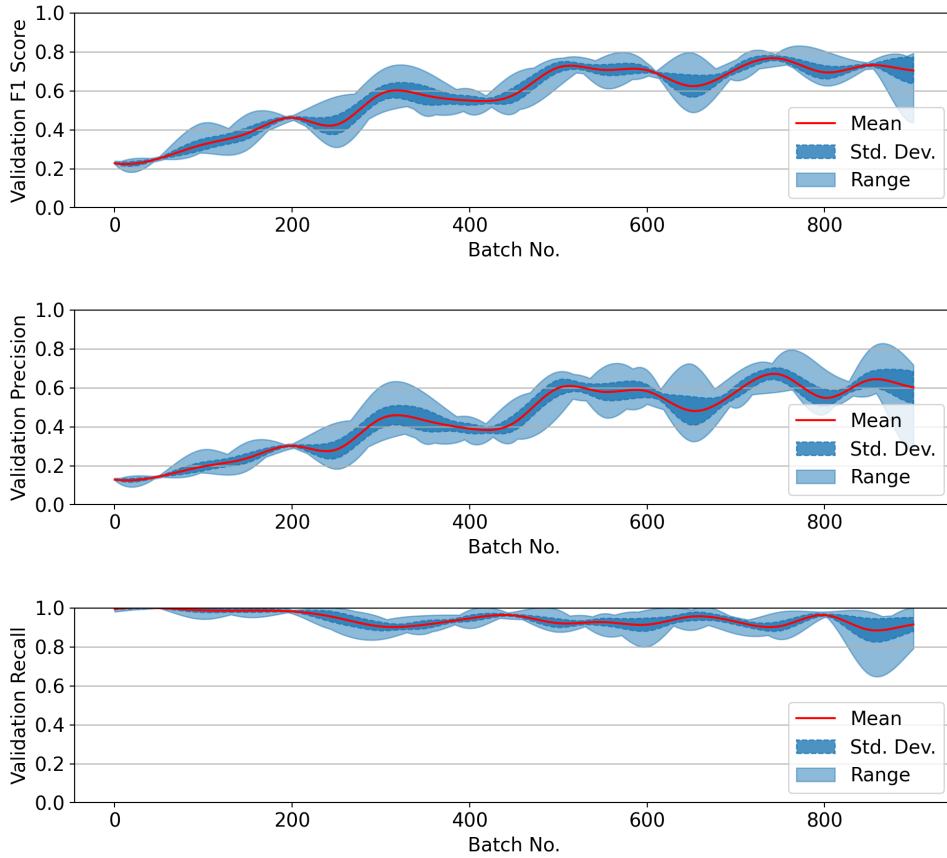
Associated with Passenger Rights

- *The train arrived over an hour late.*
- *The train was canceled and we did not arrive at the final destination, so we had to book a hotel for the night. I want a refund!*

We train the binary classifiers via 5-fold cross validation and as such, each training and evaluation procedure yields five models. The creation of the pseudo-label data is performed with averaged predictions of the five models as described in [Section 5.2](#). In [Figure 6.2](#) the variation in model training progress over the five folds is shown.



(a) Validation scores on cross validation datasets $\mathcal{D}_{val_{cv}}$

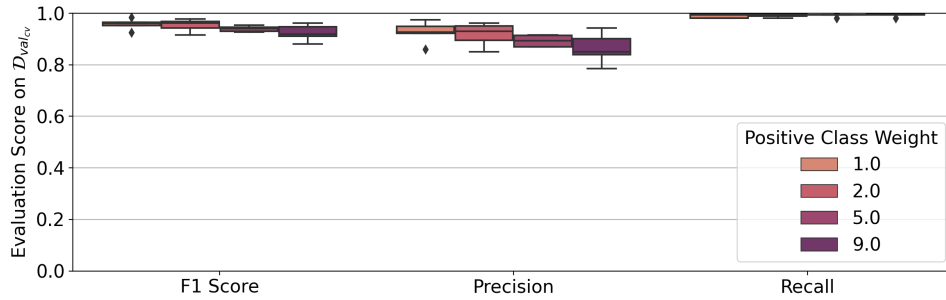


(b) Validation scores on holdout validation dataset $\mathcal{D}_{val_{ood}}$

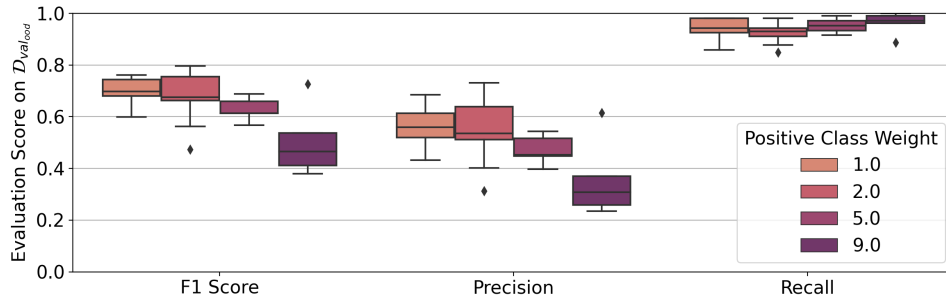
Figure 6.1: Evaluation curves over five cross validation folds on cross validation data $\mathcal{D}_{val_{cv}}$ and holdout validation data $\mathcal{D}_{val_{ood}}$

The panels show the mean F1-score, precision and recall as well as their corresponding standard deviation and range on the cross validation dataset $\mathcal{D}_{val_{cv}}$ (a) and the holdout validation dataset $\mathcal{D}_{val_{ood}}$ (b). The model precision shows the highest variance and generally the lowest value in terms of the three evaluation metrics. At the same time, the recall score is very high with values over 0.98 for both validation datasets. This validates the previously stated focus on the reduction of false positives, improving precision and consequently the F1-score.

In addition to the *Active Learning* approach to improve the model precision by curating a more informative training dataset, we try to force the model to increase the focus on positive examples by employing a positive weight parameter in the binary cross-entropy and focal loss functions. In Figure 6.2 the performance results on both validation datasets for different positive weight parameters are shown. The best model performance is reached with a balanced weight between positives and negatives.



(a) Performance on cross validation datasets $\mathcal{D}_{val_{cv}}$



(b) Performance on holdout validation dataset $\mathcal{D}_{val_{ood}}$

Figure 6.2: Influence of Positive Class Weight on model performance

We find that high positive class weights lead to a decrease in performance resulting from low precision scores. In case of high positive class weights, the model weight updates are being dominated by the positively labeled samples. Consequently, the influence of the negatively labeled data, which is richer in diversity and generally larger in samples, decreases, leading to the performance degradation. Instead of directly influencing the model weight

updates via the positive weight parameter, we employ the less invasive decision threshold tuning, which has no effect on the weight update.

Key Results

- The dataset improvement via *Active learning* combined with adversarial examples is highly effective, yielding an improvement in F1 score of $\sim 10\%$
- Employing a high positive label weight leads to a reduction in model precision, because model weight updates depend too strongly on the smaller subset of positively labeled data.

6.1.2 Multi-label classifiers

To put the performance of the multi-label LLM models into perspective we train SVM and hyperparameter-optimized XGBoost classifiers utilizing bag-of-words text representations as well as bi-directional LSTMs as baseline models. In this subsection, we compare the results of the ensemble of binary ELECTRA classifiers, the baseline multi-label models and the multi-label LLM models.

Baseline models

To provide a performance baseline we perform extensive hyperparameter tuning with XGBoost models using *bag-of-words* input representations. In Figure 6.3 we show a parallel coordinates plot containing a selection of hyperparameters and evaluation metrics corresponding to the XGBoost hyperparameter tuning procedure.

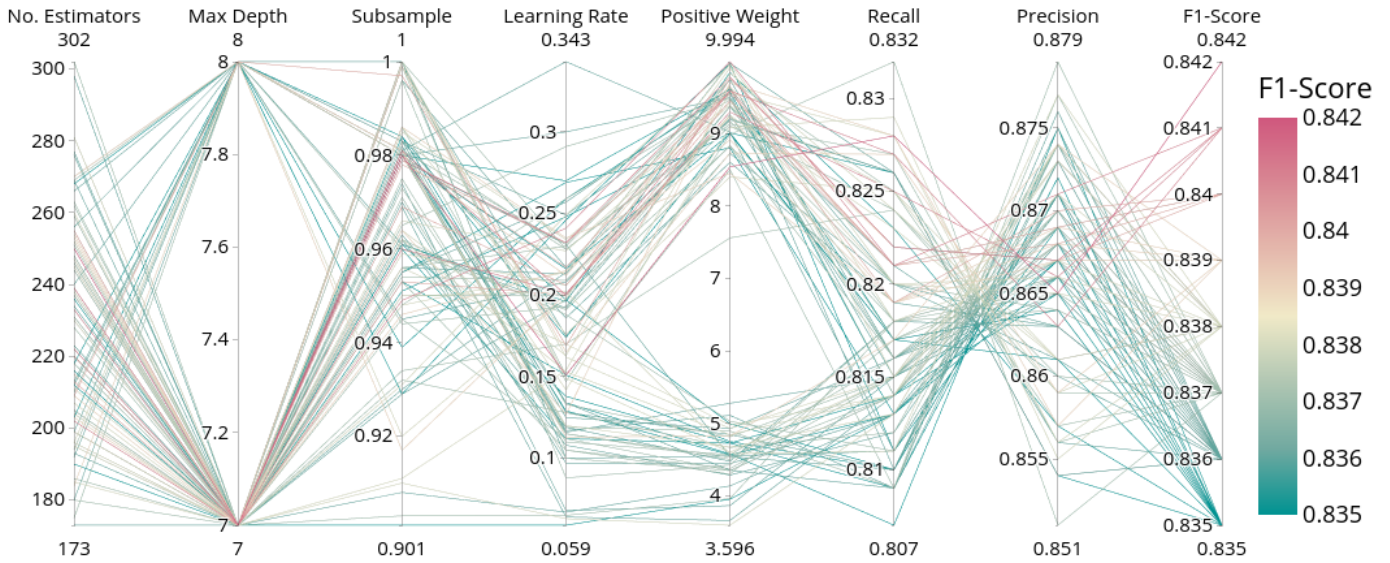


Figure 6.3: Multi-label XGBoost macro evaluation metrics on the holdout validation data $\mathcal{D}_{val,ood}$ corresponding to different hyperparameter configurations in a Parallel Coordinates Plot

In contrast to the binary [LLM](#) classifier for category *Passenger Rights* the XGBoost classifier profits from a high positive weight parameter combating the imbalanced distribution of the majority of labels. The best number of estimators and the maximum depth of the gradient boosted trees are close to the default values of 250 and 6, respectively. These values have been found to be good general configurations [\[CG16\]](#). For the optimization of the [LSTM](#) models we employ pre-trained *fastText* and *GloVe* word embeddings as well as embeddings specifically pre-trained on textual content in context of *Deutsche Bahn*. We also vary the hidden dimensions, number of feed-forward and bi-directional LSTM layers as well as the amount of dropout applied to the model. The results of the best [LSTM](#) models are incorporated in the subsequent multi-label result comparison.

Transformer based encoder models

As with the baseline [XGBoost](#) and [LSTM](#) models, we use the pseudo-label data generated from the ensemble of binary [LLMs](#) as training data. As explained in [Section 3.2.2](#) and [Section 5.2](#), we utilize early stopping and multiple evaluation datasets to detect and combat a performance degradation resulting from biased data. The [LLM](#) models show a high performance with only minimal hyperparameter tuning. The results of the best multi-label model on the test dataset are shown in [Figure 6.4](#).

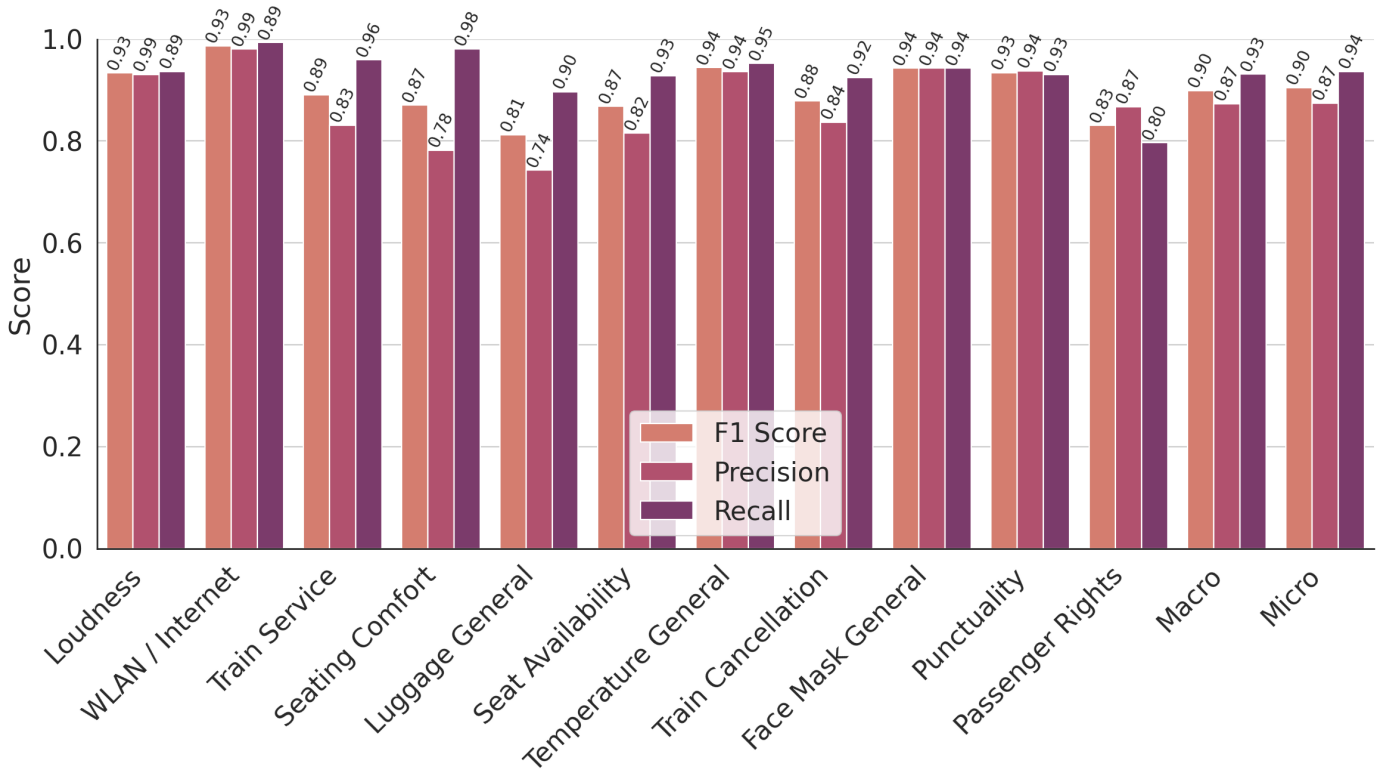


Figure 6.4: ELECTRA: F1 Score, Recall and Precision per class on test data

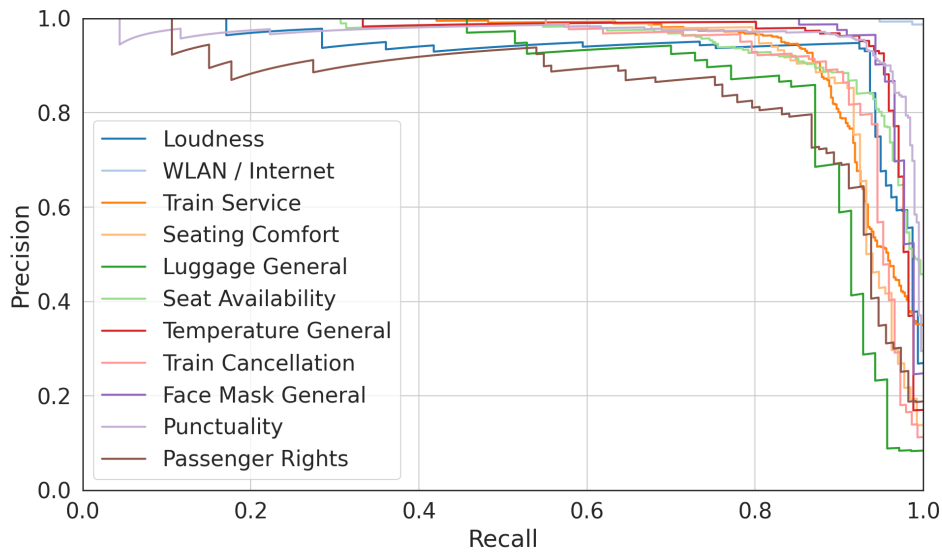


Figure 6.5: ELECTRA: Precision-Recall Curves for test data

The predictive performance in terms of F1-score is above 0.8 for all categories and for most categories close to or above 0.9 with the worst performing categories being *Luggage General* and *Passenger Rights*. For *Luggage General*

this can mostly be attributed to the contextual overlap with category *Seating Comfort*, whose corresponding samples often refer to space and furniture in a more general way than the samples corresponding to *Luggage General*, which refer to the space available for luggage. The same is true for category *Passenger Rights*, which has high overlap with category *Punctuality*, but with narrower contextual focus as described in [Section 6.1.1](#). Another contributing factor for the worse evaluation results of both categories might be the relatively low number of positives, with 70 and 113 examples of 1313 examples for *Luggage General* and *Passenger Rights*, respectively, which are among the lowest numbers of positives in the dataset. This can also be seen in the precision-recall plot shown in [Figure 6.5](#) with large vertical step sizes especially for *Luggage General* corresponding to an increase of false positives going from left to right.

In [Table 6.3](#) the predictive performance of the best models corresponding to different model architecture types on the test data are shown.

| Category | Ensemble ELECTRA | Multi-Label ELECTRA | ML 8-Bit ELECTRA | Multi-Label DistilBERT | Multi-Label LSTM | Multi-Label XGBoost | SVM |
|--------------------|---------------------|------------------------|---------------------|---------------------------|---------------------|------------------------|-------|
| Macro | 0.902 | 0.899 | 0.888 | 0.891 | 0.860 | 0.842 | 0.629 |
| Micro | 0.907 | 0.905 | 0.894 | 0.901 | 0.874 | 0.853 | 0.681 |
| Face Mask | 0.955 | 0.943 | 0.920 | 0.927 | 0.917 | 0.945 | 0.644 |
| Loudness | 0.943 | 0.933 | 0.930 | 0.924 | 0.921 | 0.888 | 0.767 |
| Luggage | 0.822 | 0.812 | 0.797 | 0.837 | 0.724 | 0.775 | 0.376 |
| Passenger Rights | 0.792 | 0.830 | 0.805 | 0.789 | 0.774 | 0.667 | 0.305 |
| Punctuality | 0.906 | 0.934 | 0.939 | 0.940 | 0.924 | 0.920 | 0.805 |
| Seat Availability | 0.897 | 0.868 | 0.838 | 0.873 | 0.833 | 0.788 | 0.632 |
| Seating Comfort | 0.863 | 0.870 | 0.856 | 0.810 | 0.756 | 0.711 | 0.539 |
| Temperature | 0.936 | 0.944 | 0.937 | 0.942 | 0.938 | 0.912 | 0.745 |
| Train Cancellation | 0.916 | 0.879 | 0.877 | 0.857 | 0.825 | 0.807 | 0.571 |
| Train Service | 0.906 | 0.891 | 0.876 | 0.900 | 0.882 | 0.869 | 0.641 |
| WLAN / Internet | 0.990 | 0.987 | 0.990 | 0.997 | 0.961 | 0.984 | 0.892 |

Table 6.3: Comparison of the predictive performance of different model architectures on the test data via F1-Score

The evaluation results offer multiple interesting perspectives. For one, we observe that the **LLM** models on average perform significantly better than the optimized baseline **XGBoost** models. The **LSTM** baseline models also perform worse than the **LLM** models, although by a lesser degree. We find that both baseline architectures show significant performance drops on specific categories. The 8-bit quantization slightly impacts the performance of the **LLM** models with a decrease of 0.01 in F1-score on average. At the same time the model size can be compressed to a fourth of its original size. The pruned and distilled **DistilBERT** transformer variant shows competitive results with the much larger **ELECTRA** and even outperforms the larger model on some of the categories like *Luggage General* and *Train Service*. This could be the result of the regularizing effect of a lower capacity model.

Key Results

- Compressing an eleven category ensemble of **ELECTRA** models into a single multi-label **ELECTRA** model is possible without significant performance degradation.
- We observe a predictive performance improvement with categories that profit from label correlations not accessible to binary models.
- The compressed models, *DistilBERT* and 8-bit quantized **ELECTRA**, show a reduction in F1 score of 1.5%.
- **LSTM** and **XGBoost** baseline models show a more significant reduction in predictive performance with 4.7% and 6.7%, respectively.
- All models significantly improve on the baseline set by the **SVM** classifier.

6.1.3 Scaling to higher numbers of categories

We extend the category selection to a total of 82 categories to investigate the scalability of the presented workflow. We train 82 binary **ELECTRA** models via 5-fold cross validation on expert-annotated binary data. With the trained models we extend the previously used 11-category pseudo-label dataset, consisting of 37621 samples, with pseudo-labels for all 82 categories. Following, we train multi-label **LLMs** on the pseudo-label training data and choose to evaluate both binary and multi-label models on a fixed cross-validation fold of each binary expert-annotated dataset, since manually multi-label annotating 82 categories on the holdout validation set is too time consuming. In [Figure 6.6](#) an overview of the F1-score results (red) and the label distribution of the training dataset (blue) is shown for the binary classifiers, the multi-label classifiers as well as the difference between the binary and multi-label

results. The categories are sorted by the F1-score of the binary classifiers in descending order and keep their ordering in all three panels.

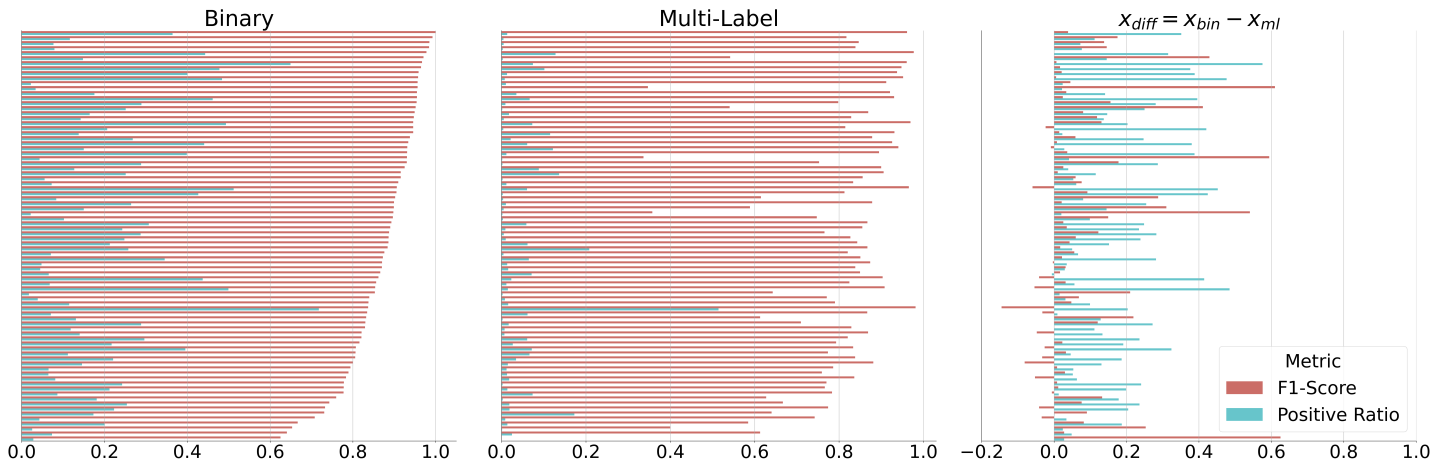


Figure 6.6: F1-Scores and Positive Label Training Data Ratio in context of the binary classifiers, the multi-label classifier and respective difference

A very noticeable difference is the much lower ratio of positives in the pseudo-label data used to train the multi-label data. This can be attributed to the sampling scheme used to create the binary-annotated training data, in which acquiring a representative amount of positives is preferred over faithfully capturing the true population distribution, which might display a very low positive ratio. The pseudo-label data on the other hand was created by predicting the label associations of an i.i.d. unlabeled dataset and as such might be closer to the true label distribution. Some of the categories show a very noticeable F1-score performance degradation between the results of the binary and multi-label classifiers. We assume that the change in label distribution might be a major contributing factor. A linear regression with the logarithmic ratio between the binary and the multi-label label distribution as independent variable and the difference in F1-score as dependent variable confirms this assumption. In Figure 6.7 the true difference in F1-score y as well as the fitted values \hat{y} and respective prediction intervals are plotted against the logarithmic ratio of label distributions. With the prediction interval incorporating the uncertainty of the estimated parameters as well as the error term ϵ .

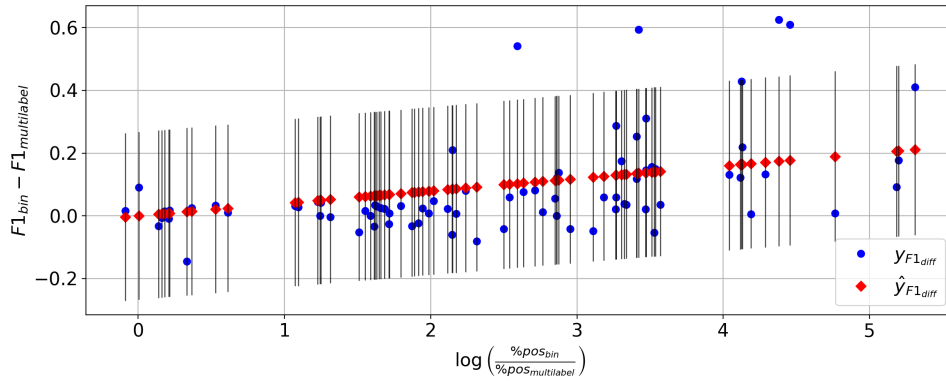


Figure 6.7: Linear regression plot showing the true values y and fitted values \hat{y} corresponding to the difference in F1-Score between binary and multi-label models on the binary cross-validation data against the dependent variable x corresponding to the logarithmic ratio of the binary and multi-label label distributions.

In the linear regression summary shown in [Table 6.4](#) we can see that with an $R^2 = 0.406$, the linear model is able to explain around 40% of the variance of the dependent variable $F_{1_{diff}}$. Consequently, there must be other factors contributing to the variance of $F_{1_{diff}}$ that are not included in the univariate linear model.

| | | | |
|--------------------------|----------------|-------------------------------------|----------|
| Dep. Variable: | $F_{1_{diff}}$ | R-squared (uncentered): | 0.406 |
| Model: | OLS | Adj. R-squared (uncentered): | 0.399 |
| Method: | Least Squares | F-statistic: | 55.35 |
| Df Residuals: | 81 | Prob (F-statistic): | 9.44e-11 |
| No. Observations: | 82 | Log-Likelihood: | 48.911 |
| Df Model: | 1 | AIC: | -95.82 |
| Covariance Type: | nonrobust | BIC: | -93.41 |

| | coef | std err | t | P> t | [0.025 | 0.975] |
|-----------------------|--------|---------|-------|-------|--------|--------|
| log(pos_ratio) | 0.0398 | 0.005 | 7.440 | 0.000 | 0.029 | 0.050 |

Table 6.4: OLS Regression Results

Still, a significant influence can be attributed to the dependent variable considering the high t -statistic and its respective small p-value. We find that most of the average performance degradation results from a small set of categories, as can be seen in [Table 6.5](#). Here we remove the categories, on which the multi-label classifier performs the worst and observe that after removing 16 of 82 categories, the difference in macro F1-score shrinks from a difference of 0.08 to a difference of just 0.03.

| No. of categories | Macro $F1_{bin}$ | Macro $F1_{ML}$ |
|-------------------|------------------|-----------------|
| 82 | 0.870 | 0.786 |
| 81 | 0.873 | 0.796 |
| 80 | 0.873 | 0.801 |
| 79 | 0.872 | 0.808 |
| 78 | 0.871 | 0.813 |
| 77 | 0.874 | 0.819 |
| 76 | 0.873 | 0.822 |
| ... | ... | ... |
| 66 | 0.884 | 0.854 |

Table 6.5: Macro F1-scores of binary and multi-label classifiers by choosing the best performing n categories for the multi-label model

Considering that we performed no additional steps to optimize the 82-category pseudo-label dataset, we trained the multi-label classifier on, this is an impressive result. With additional dataset curation and focus on the pseudo-labels corresponding to the weakly performing categories, it might be possible to achieve competitive results on all 82 categories. Unfortunately, such an investigation was out of scope for this work. Nonetheless, the above results show that by taking an average performance hit of 0.03 in macro F1-score, we can compress 66 binary LLMs into a single multi-label model.

Key Results

- The compressed 82 category multi-label model only shows a reduction in performance of 3% on over 75% of the categories compared to the binary ensemble without any additional data curation.
- The multi-label model shows a significant drop in predictive performance on a small subset of categories.
- The sparsity of positive labels in the generated pseudo-label data negatively impacts the performance of the compressed multi-label model.

6.2 INFERENCE LATENCY OPTIMIZATION ON CPU

In this section we present the results of our latency optimization experiments. As initially stated, our proposed workflow encompasses the knowledge distillation of binary classifier ensembles into single multi-label models. In the previous section we compared the predictive performance of the binary ensemble with several multi-label model architectures. In this section we compare the inference latencies of the proposed baseline *XGBoost* and *LSTM*

models and the LLM models. We convert the PyTorch models, *LSTMs* and *LLMs*, into an optimized Open Neural Network Exchange (*ONNX*) graph representation. The optimization procedure entails the reduction of redundant nodes, constant folding, which is the pre-computation of static parts of the network graph as well as node fusion, in which multiple nodes of the graph are combined. This is for example the combination of the single operations of the layer normalization and corresponding skip-connection in the transformer architecture into a single node. We finally transform the optimized *ONNX* model graph into a dynamically quantized 8-bit integer representation. The measurements are performed with a warmup of 100 repetitions, which are discarded, and subsequent 300 measurements which are used to evaluate the latency performance of the models on the CPU hardware and environment specified in Section 5.3. In Figure 6.8 the measured inference latencies of the baseline *XGBoost* models for input data with different sequence lengths are visualized as boxplots. The median inference latencies for all sequence lengths are around 30ms, which is significantly below the constraint of 100ms. The *XGBoost* model utilizes *bag-of-words* text representations for the input documents which means that only the preprocessing step is affected by higher sequence lengths. The input to the algorithm itself is a vector of vocabulary length $|V|$ and as such the measured latencies show no significant difference regarding different sequence lengths.

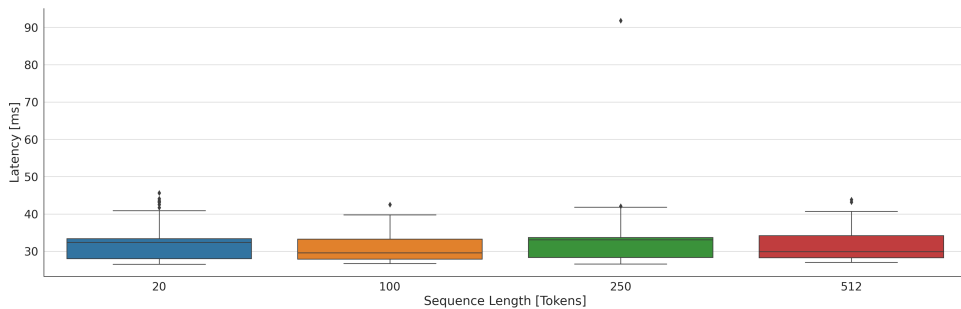


Figure 6.8: Per-Sample Latencies for XGBoost baseline models and different input sequence lengths

In the three subfigures of Figure 6.9 we show the measured per-sample latencies for the deep learning models for different sequence lengths. The three compared model architectures are the baseline bi-directional *LSTM* and the two LLMs, *DistilBERT* and *ELECTRA*. Each subfigure consists of three columns with each column representing one of the three optimization levels *PyTorch* (unoptimized), *ONNX* (graph-optimized) and *ONNX 8-bit Quantized* (graph-optimized and quantized). A relatively unsurprising observation is the correlation between model size (and thus number of required mathematical operations) and inference latency. In all of the experiments the *ELECTRA* model, having twice the number of encoder-layers of *DistilBERT*, shows the highest inference latency.

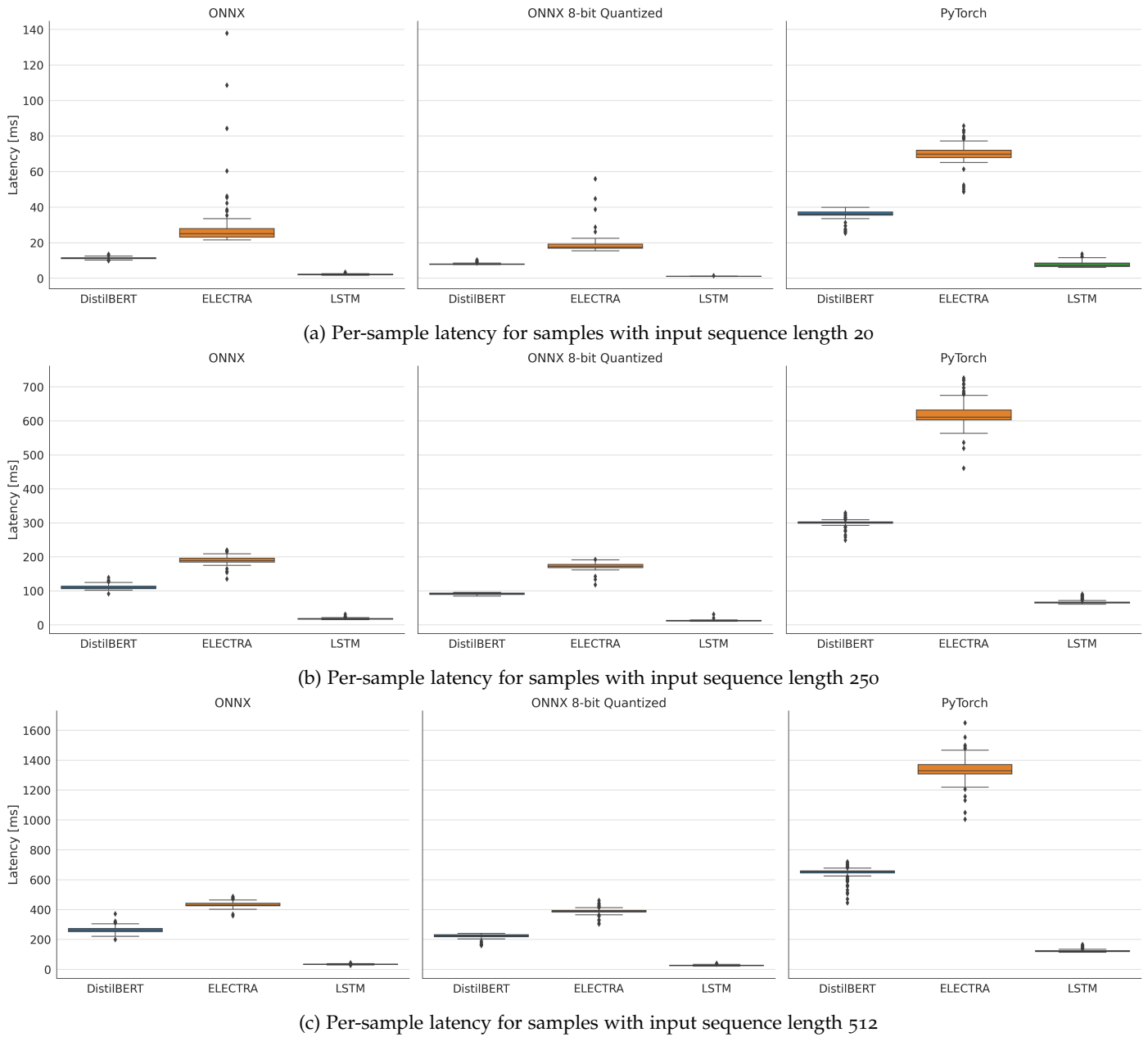


Figure 6.9: Per-sample latency for different model architectures and sequence lengths

The multi-label [LSTM](#) consistently shows the lowest inference latencies given the much lower amount of trainable parameters, which amount to approximately 5 million parameters in comparison to the 66 million parameters in case of *DistilBERT* and 110 million parameters in case of *ELECTRA*. For all models the biggest improvement in inference latency can be achieved by the transformation to a graph-optimized *ONNX*-format. For sequence lengths of 250 tokens all model types reach mean inference latencies below 200ms as can be seen in more detail in [Table 6.6](#).

| Model Name | Execution Type | \bar{t} | t_{std} | $t_{0.5}$ | $t_{0.99}$ |
|------------|----------------------|-----------|-----------|-----------|------------|
| DistilBERT | ONNX | 111.40 | 5.99 | 110.36 | 129.65 |
| | ONNX 8-bit Quantized | 92.17 | 2.30 | 92.84 | 95.71 |
| | PyTorch | 301.18 | 8.73 | 301.64 | 324.83 |
| ELECTRA | ONNX | 190.53 | 9.92 | 189.70 | 218.07 |
| | ONNX 8-bit Quantized | 173.77 | 8.90 | 173.05 | 189.91 |
| | PyTorch | 620.76 | 33.18 | 611.44 | 718.98 |
| LSTM | ONNX | 18.45 | 1.83 | 18.21 | 25.01 |
| | ONNX 8-bit Quantized | 12.72 | 1.59 | 12.30 | 15.22 |
| | PyTorch | 67.29 | 5.44 | 65.90 | 88.64 |
| XGBoost | sklearn | 31.87 | 4.70 | 33.11 | 41.45 |

Table 6.6: Per-sample latency sequence length 250

| Model Name | Execution Type | \bar{t} | t_{std} | $t_{0.5}$ | $t_{0.99}$ |
|------------|----------------------|-----------|-----------|-----------|------------|
| DistilBERT | ONNX | 266.07 | 20.26 | 263.36 | 318.54 |
| | ONNX 8-bit Quantized | 223.63 | 13.97 | 224.42 | 239.39 |
| | PyTorch | 649.17 | 33.77 | 655.02 | 709.40 |
| ELECTRA | ONNX | 434.46 | 16.49 | 431.24 | 482.03 |
| | ONNX 8-bit Quantized | 389.61 | 17.05 | 389.07 | 437.16 |
| | PyTorch | 1338.78 | 67.02 | 1330.71 | 1501.26 |
| LSTM | ONNX | 34.98 | 2.35 | 34.95 | 41.26 |
| | ONNX 8-bit Quantized | 26.96 | 2.61 | 26.66 | 33.41 |
| | PyTorch | 124.95 | 8.92 | 121.93 | 158.73 |
| XGBoost | sklearn | 31.46 | 3.45 | 29.94 | 40.57 |

Table 6.7: Per-sample latency sequence length 512

Utilizing the full sequence length of 512 tokens processable by *ELECTRA* and *DistilBERT* results in average inference latencies much higher than the 100ms constraint we aimed at, as can be seen in [Table 6.7](#). We find that through 8-bit quantization an additional latency decrease of up to 31% in case of the *LSTM* models can be reached with the benefit of a 75% reduction in model size. In comparison to the unoptimized *PyTorch* variants, models executed in an *ONNX* runtime environment show much lower variance in inference latency.

Key Results

- Reducing the inference latency of transformer models to sub 100ms on the specified commodity hardware is possible by employing model compression techniques, reduced maximum input sequence lengths as well as a dedicated execution environment like *ONNX runtime*.
- *XGBoost* and *LSTM* models perform inference a magnitude faster than the corresponding transformer models.

DISCUSSION

In this master's thesis we proposed an end-to-end process for the creation of real-time-capable multi-label LLM classifiers based on disjunct binary-labeled data. We specifically investigated three research questions that are subsequently discussed in detail.

Is it possible to train a performant multi-label model without expert-annotated multi-label data?

As first and arguably most important question regarding the feasibility of our proposed workflow, we investigated if it is possible to create competitive multi-label models through knowledge distillation with pseudo-labels. In [Section 6.1.2](#) we compared the performance of such multi-label model architectures with the ensemble of optimized binary LLM classifiers used to generate the pseudo-label training data. Given that [SSL](#) approaches typically incorporate small amounts of expert-labeled data as explained in [Chapter 2](#), we assumed that our approach, solely relying on pseudo-label data, should lead to a noticeable performance degradation. Interestingly, this is not the case in context of the 11 category experiment, where the compressed multi-label model performs on-par with the binary ensemble. We even see a performance increase in the prediction of categories for which pairwise label correlations in the dataset exist. This is most prominent for category *Passenger Rights*, on which the multi-label *ELECTRA* model achieves the best performance across all included model architectures. We presume that pairwise label correlations between *Passenger Rights* and the thematically adjacent categories *Punctuality* and *Train Cancellation*, as shown in [Figure 5.5](#), help the model in its decision process and lead to the increased performance.

In the scalability experiment in [Section 6.1.3](#) we have shown, that our proposed workflow is scalable to much higher numbers of categories. Additionally, it has to be noted that we performed no additional data optimization for the extension to 82 categories. Due to resource and time constraints, we performed the pseudo-labeling for the additional 71 categories exclusively on the 37,000 samples optimized for the eleven category experiment. Under this consideration, the experiment worked surprisingly well. The multi-label model performed competitively with the binary models on over 75% of categories. However, the multi-label model shows noticeable performance degradation on 16 of the 82 categories. Unfortunately, creating pseudo-label data for such a high number of categories via cross-validation and multiple random seeds is very time-consuming and was out of scope for this work. It would, however, be a worthwhile endeavour to focus on improving the

multi-label performance on the worse performing 16 categories. Another potentially viable extension of the experimental setup is the pseudo-labeling of a much higher amount of unlabeled data. This would provide higher degrees of freedom in selecting a viable training data subset, integrating all 82 categories into the selection process.

Can a multi-label language model outperform multiple binary language models and thus, can the multi-label model implicitly use label correlations to its advantage?

As discussed in [Chapter 2](#) and [Section 3.3](#), pre-trained *transformer*-based language models are highly suitable for transfer learning. As such, we hypothesized the multi-label models to improve on the performance of binary models, where knowledge transfer, specifically label correlations, could be leveraged. We have seen that the multi-label model shows significant improvement in F1-score with respect to category *Passenger Rights*, which is part of the two most highly correlated label pairs. However, most of the eleven categories show relatively low absolute pairwise correlations and as such there is only sparse shared structure the multi-label model can potentially exploit to improve on the binary classifiers. Even though the most correlated category *Passenger Rights* experienced a significant boost in performance, the multi-label model performed slightly worse on most of the remaining categories. To pass a more confident verdict regarding the influence of label correlation on the multi-label models performance, future works with a more extensive investigation into categories with higher pairwise correlation would be advantageous.

Can a multi-label LLM reach an inference latency of sub 100ms on CPU hardware?

We showed that it is possible to distill the knowledge of an ensemble of binary classifiers into a single multi-label model. After transforming the original deep learning models into an [ONNX](#) graph representation, we optimized the graph with the removal of redundant nodes, fusion of adjacent nodes as well as dynamic quantization. With the smaller *DistilBERT* and sequence lengths of 250 tokens, we find that it is possible to reach inference latencies under 100ms on commodity CPU hardware. This meets our expectations as discussed in [Chapter 2](#). We could not, however, satisfy this constraint without reducing the maximum input sequence length. As discussed in [Section 3.2](#), the number of tokens contained in the individual feedbacks are generally less than 100. Still, reducing the maximum sequence length too much, leads to an enormous information loss when handling significantly longer documents. To further optimize the inference latency, a simple but costly approach might be the utilization of more performant CPUs. A possible topic for future research might be the incorporation of inference engines capable of executing sparse neural network graphs.

7.1 CONCLUSION

As discussed in [Chapter 7](#), we could successfully reach the overarching goal of creating an end-to-end workflow ultimately compressing a binary ensemble into a multi-label model exclusively utilizing binary-annotated training data by means of intermediary pseudo-labels. In addition, we could show that the resulting multi-label model reaches competitive predictive performance compared to the teacher ensemble model. Through the application of different latency optimization techniques like neural network graph optimization and quantization, we could also show that a compressed and optimized multi-label model can reach real-time inference latencies below 100ms per-sample. However, we have seen that the scalability of the proposed workflow needs closer inspection as the predictive performance degradation of the multi-label model in the scalability experiment in [Section 6.1.3](#) has shown.

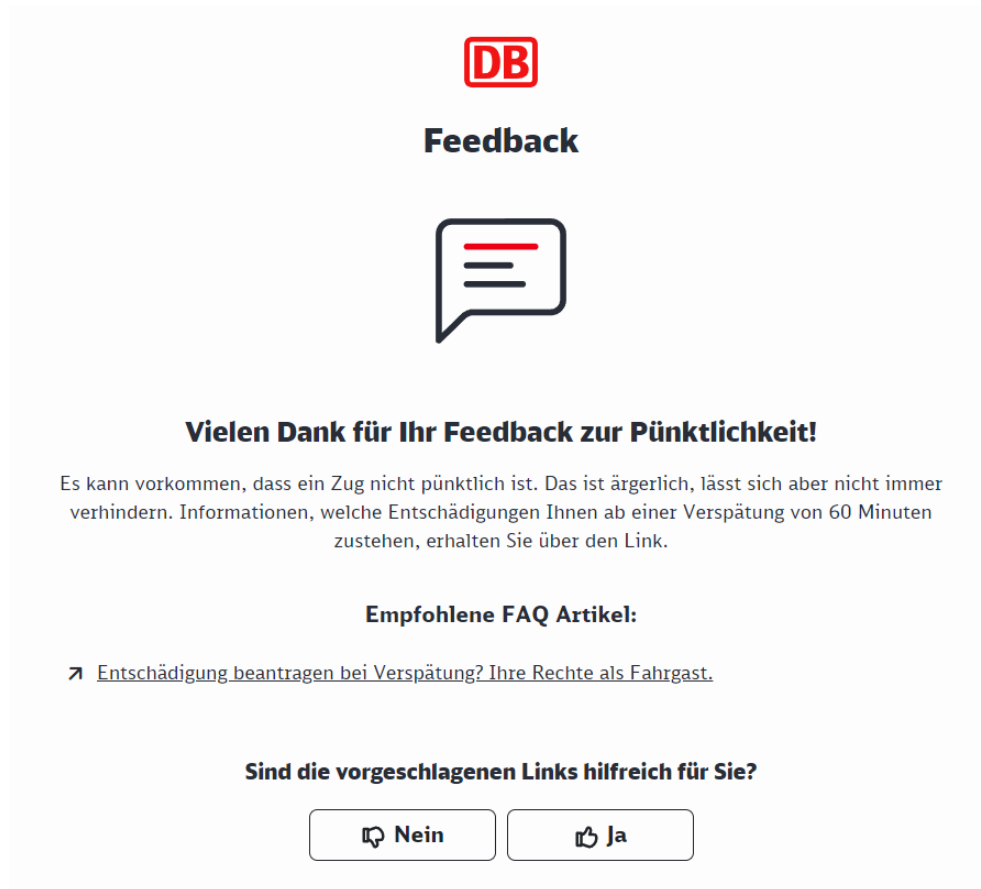
Weaknesses and future work

In [Chapter 7](#) we outlined that our proposed workflow could successfully be implemented in an eleven category study. The extension to 82 categories however showed significant performance degradation on 16 of the 82 categories. As discussed in [Section 3.2.2](#), the intermediary pseudo-labels carry an inherent bias that follows from the specific model architecture of the generating model as well as the bias introduced from the annotation procedure used to curate the binary training data. It might be worthwhile to investigate the performance of models trained on pseudo-label data with specific focus on varying levels of linguistic or thematic difficulty. Improving the selection of training data for an extended multi-label model could be another subject for subsequent work. As previously stated, first steps could be the creation of much higher amounts of pseudo-labeled data. Additionally, in this work we restricted ourselves to the selection of informative pseudo-label data by means of *Uncertainty Filtering* setting a hard threshold for each individual label. It could be helpful to incorporate pre-clustering [[NS04](#)] or a hierarchical label taxonomy [[DHo8](#)] into the sampling process to acquire a more informative set of pseudo-label training data. Also, in terms of inference latency optimization, execution environments have emerged enabling a hardware-efficient execution of sparse LLMs in addition to quantization and network graph optimization.

Practical Application of the Thesis Results

Concluding this work, we are proud to mention that we could successfully integrate the proposed workflow into the feedback process at *Deutsche Bahn*. A model based on the findings of this work is currently live and accessible to a potential 200 million customers. Since its deployment on March 16th, 2023 the model has already served over 40,000 customers by enabling an

instant response to the customers feedback with contextualized information. In [Figure 7.1](#) the live response to a customers feedback regarding category *Punctuality* is shown.



The image shows a screenshot of a feedback response from Deutsche Bahn (DB). At the top, the DB logo is displayed in red. Below it, the word "Feedback" is written in bold black text. A speech bubble icon with three horizontal lines inside is centered. The main heading reads "Vielen Dank für Ihr Feedback zur Pünktlichkeit!". The body text explains that train delays are frustrating but not always preventable, and offers information on compensation for delays of 60 minutes or more via a link. Below this, a section titled "Empfohlene FAQ Artikel:" lists a link: "➤ [Entschädigung beantragen bei Verspätung? Ihre Rechte als Fahrgast.](#)". At the bottom, a question asks "Sind die vorgeschlagenen Links hilfreich für Sie?" with two buttons: "Nein" (with a thumbs-down icon) and "Ja" (with a thumbs-up icon).

Figure 7.1: Contextualized response based on the detection of category *Punctuality*

In conclusion, the results of this work serve as yet another step to the enhancement of the travel experience with *Deutsche Bahn* through machine learning.

BIBLIOGRAPHY

- [Ala18] Jay Alammam. *The Illustrated Transformer*. <http://jalammar.github.io/illustrated-transformer>. 2018. URL: <http://jalammar.github.io/illustrated-transformer>.
- [BAP14] Philip Bachman, Ouais Alsharif, and Doina Precup. “Learning with pseudo-ensembles.” In: *Advances in neural information processing systems 27* (2014).
- [BCB14] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. 2014. DOI: [10.48550/ARXIV.1409.0473](https://doi.org/10.48550/ARXIV.1409.0473). URL: <https://arxiv.org/abs/1409.0473>.
- [BH20] Maximiliana Behnke and Kenneth Heafield. “Losing Heads in the Lottery: Pruning Transformer Attention in Neural Machine Translation.” In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Online: Association for Computational Linguistics, Nov. 2020, pp. 2664–2674. DOI: [10.18653/v1/2020.emnlp-main.211](https://doi.org/10.18653/v1/2020.emnlp-main.211). URL: <https://aclanthology.org/2020.emnlp-main.211>.
- [Ber+11] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. “Algorithms for hyper-parameter optimization.” In: *Advances in neural information processing systems 24* (2011).
- [Bis06] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006. ISBN: 0387310738.
- [BM98] Avrim Blum and Tom Mitchell. “Combining labeled and unlabeled data with co-training.” In: *Proceedings of the eleventh annual conference on Computational learning theory*. 1998, pp. 92–100.
- [Bro+20] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. “Language models are few-shot learners.” In: *Advances in neural information processing systems 33* (2020), pp. 1877–1901.
- [BCNM06] Cristian Bucila, Rich Caruana, and Alexandru Niculescu-Mizil. “Model Compression.” In: *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. KDD '06*. Philadelphia, PA, USA: Association for Computing Machinery, 2006, 535–541. ISBN: 1595933395. DOI: [10.1145/1150402.1150464](https://doi.org/10.1145/1150402.1150464). URL: <https://doi.org/10.1145/1150402.1150464>.

- [CG16] Tianqi Chen and Carlos Guestrin. “XGBoost: A Scalable Tree Boosting System.” In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’16. San Francisco, California, USA: Association for Computing Machinery, 2016, 785–794. ISBN: 9781450342322. DOI: [10.1145/2939672.2939785](https://doi.org/10.1145/2939672.2939785). URL: <https://doi.org/10.1145/2939672.2939785>.
- [Cla+19] Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D. Manning. “What Does BERT Look at? An Analysis of BERT’s Attention.” In: *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*. Florence, Italy: Association for Computational Linguistics, Aug. 2019, pp. 276–286. DOI: [10.18653/v1/W19-4828](https://aclanthology.org/W19-4828). URL: <https://aclanthology.org/W19-4828>.
- [Cla+20] Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. *ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators*. 2020. DOI: [10.48550/ARXIV.2003.10555](https://arxiv.org/abs/2003.10555). URL: <https://arxiv.org/abs/2003.10555>.
- [CV95] Corinna Cortes and Vladimir Vapnik. “Support-vector networks.” In: *Machine learning* 20 (1995), pp. 273–297.
- [CB16] Matthieu Courbariaux and Yoshua Bengio. “BinaryNet: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1.” In: *ArXiv abs/1602.02830* (2016).
- [DH08] Sanjoy Dasgupta and Daniel Hsu. “Hierarchical sampling for active learning.” In: *Proceedings of the 25th international conference on Machine learning*. 2008, pp. 208–215.
- [DCH10] Krzysztof Dembczynski, Weiwei Cheng, and Eyke Hüllermeier. “Bayes Optimal Multilabel Classification via Probabilistic Classifier Chains.” In: *International Conference on Machine Learning*. 2010.
- [Dem+10] Krzysztof Dembczynski, Willem Waegeman, Weiwei Cheng, and Eyke Hüllermeier. “Regret Analysis for Performance Metrics in Multi-Label Classification: The Case of Hamming and Subset Zero-One Loss.” In: *ECML/PKDD*. 2010.
- [DWH12] Krzysztof Dembczyński, Willem Waegeman, and Eyke Hüllermeier. “An Analysis of Chaining in Multi-Label Classification.” In: *Proceedings of the 20th European Conference on Artificial Intelligence*. ECAI’12. Montpellier, France: IOS Press, 2012, 294–299. ISBN: 9781614990970.
- [Dem+12] Krzysztof Dembczyński, Willem Waegeman, Weiwei Cheng, and Eyke Hüllermeier. “On label dependence and loss minimization in multi-label classification.” In: *Machine Learning* 88 (July 2012). DOI: [10.1007/s10994-012-5285-8](https://doi.org/10.1007/s10994-012-5285-8).

- [Den+20] Lei Deng, Guoqi Li, Song Han, Luping Shi, and Yuan Xie. "Model Compression and Hardware Acceleration for Neural Networks: A Comprehensive Survey." In: *Proceedings of the IEEE* 108.4 (2020), pp. 485–532. DOI: [10.1109/JPROC.2020.2976475](https://doi.org/10.1109/JPROC.2020.2976475).
- [Dev+18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2018. DOI: [10.48550/ARXIV.1810.04805](https://doi.org/10.48550/ARXIV.1810.04805). URL: <https://arxiv.org/abs/1810.04805>.
- [Goo+14] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. *Generative Adversarial Networks*. 2014. DOI: [10.48550/ARXIV.1406.2661](https://doi.org/10.48550/ARXIV.1406.2661). URL: <https://arxiv.org/abs/1406.2661>.
- [GDA20] Mitchell A. Gordon, Kevin Duh, and Nicholas Andrews. *Compressing BERT: Studying the Effects of Weight Pruning on Transfer Learning*. 2020. DOI: [10.48550/ARXIV.2002.08307](https://doi.org/10.48550/ARXIV.2002.08307). URL: <https://arxiv.org/abs/2002.08307>.
- [Gou+21] Jianping Gou, Baosheng Yu, Stephen J. Maybank, and Dacheng Tao. "Knowledge Distillation: A Survey." In: *International Journal of Computer Vision* 129.6 (2021), pp. 1789–1819. DOI: [10.1007/s11263-021-01453-z](https://doi.org/10.1007/s11263-021-01453-z). URL: <https://doi.org/10.1007/s11263-021-01453-z>.
- [Guo+17] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. "On calibration of modern neural networks." In: *International conference on machine learning*. PMLR. 2017, pp. 1321–1330.
- [Gup+15] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. "Deep learning with limited numerical precision." In: *International conference on machine learning*. PMLR. 2015, pp. 1737–1746.
- [Han+16] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A. Horowitz, and William J. Dally. "EIE: Efficient Inference Engine on Compressed Deep Neural Network." In: *Proceedings of the 43rd International Symposium on Computer Architecture*. ISCA '16. Seoul, Republic of Korea: IEEE Press, 2016, 243–254. ISBN: 9781467389471. DOI: [10.1109/ISCA.2016.30](https://doi.org/10.1109/ISCA.2016.30). URL: <https://doi.org/10.1109/ISCA.2016.30>.
- [He+15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. *Deep Residual Learning for Image Recognition*. 2015. DOI: [10.48550/ARXIV.1512.03385](https://doi.org/10.48550/ARXIV.1512.03385). URL: <https://arxiv.org/abs/1512.03385>.
- [HVD15] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. *Distilling the Knowledge in a Neural Network*. 2015. DOI: [10.48550/ARXIV.1503.02531](https://doi.org/10.48550/ARXIV.1503.02531). URL: <https://arxiv.org/abs/1503.02531>.

- [HS97] Sepp Hochreiter and Jürgen Schmidhuber. “Long Short-Term Memory.” In: *Neural Comput.* 9.8 (1997), 1735–1780. ISSN: 0899-7667. DOI: [10.1162/neco.1997.9.8.1735](https://doi.org/10.1162/neco.1997.9.8.1735). URL: <https://doi.org/10.1162/neco.1997.9.8.1735>.
- [Hoe+21] Torsten Hoefer, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. *Sparsity in Deep Learning: Pruning and growth for efficient inference and training in neural networks*. 2021. DOI: [10.48550/ARXIV.2102.00554](https://arxiv.org/abs/2102.48550). URL: <https://arxiv.org/abs/2102.00554>.
- [Hua+21] Kai Huang, Jie Geng, Wen Jiang, Xinyang Deng, and Zhe Xu. “Pseudo-loss confidence metric for semi-supervised few-shot learning.” In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 8671–8680.
- [HS14] Kyuyeon Hwang and Wonyong Sung. “Fixed-point feedforward deep neural network design using weights+ 1, 0, and- 1.” In: *2014 IEEE Workshop on Signal Processing Systems (SiPS)*. IEEE, 2014, pp. 1–6.
- [Jac+18] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. “Quantization and training of neural networks for efficient integer-arithmetic-only inference.” In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 2704–2713.
- [JM09] Dan Jurafsky and James H. Martin. *Speech and language processing : an introduction to natural language processing, computational linguistics, and speech recognition*. Upper Saddle River, N.J.: Pearson Prentice Hall, 2009. ISBN: 9780131873216 0131873210. URL: http://www.amazon.com/Speech-Language-Processing-2nd-Edition/dp/0131873210/ref=pd_bxgy_b_img_y.
- [KB13] Nal Kalchbrenner and Phil Blunsom. “Recurrent continuous translation models.” In: *Proceedings of the 2013 conference on empirical methods in natural language processing*. 2013, pp. 1700–1709.
- [Kim+21] Sehoon Kim, Amir Gholami, Zhewei Yao, Michael W. Mahoney, and Kurt Keutzer. “I-BERT: Integer-only BERT Quantization.” In: *Proceedings of the 38th International Conference on Machine Learning*. Ed. by Marina Meila and Tong Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, 2021, pp. 5506–5518. URL: <https://proceedings.mlr.press/v139/kim21d.html>.
- [KB14] Diederik Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization.” In: *International Conference on Learning Representations (Dec. 2014)*.

- [Lag+21] François Lagunas, Ella Charlaix, Victor Sanh, and Alexander M. Rush. *Block Pruning For Faster Transformers*. 2021. DOI: [10.48550/ARXIV.2109.04838](https://doi.org/10.48550/ARXIV.2109.04838). URL: <https://arxiv.org/abs/2109.04838>.
- [Lat+21] Chris Lattner, Mehdi Amini, Uday Bondhugula, Albert Cohen, Andy Davis, Jacques Pienaar, River Riddle, Tatiana Shpeisman, Nicolas Vasilache, and Oleksandr Zinenko. “MLIR: Scaling compiler infrastructure for domain specific computation.” In: *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. IEEE. 2021, pp. 2–14.
- [Lee+13] Dong-Hyun Lee et al. “Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks.” In: *Workshop on challenges in representation learning, ICML*. Vol. 3. 2. 2013, p. 896.
- [LG94] David D. Lewis and William A. Gale. *A Sequential Algorithm for Training Text Classifiers*. 1994. DOI: [10.48550/ARXIV.CMP-LG/9407020](https://doi.org/10.48550/ARXIV.CMP-LG/9407020). URL: <https://arxiv.org/abs/cmp-lg/9407020>.
- [Lin+17] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. *Focal Loss for Dense Object Detection*. 2017. DOI: [10.48550/ARXIV.1708.02002](https://doi.org/10.48550/ARXIV.1708.02002). URL: <https://arxiv.org/abs/1708.02002>.
- [Liu+17] Jingzhou Liu, Wei-Cheng Chang, Yuexin Wu, and Yiming Yang. “Deep learning for extreme multi-label text classification.” In: *Proceedings of the 40th international ACM SIGIR conference on research and development in information retrieval*. 2017, pp. 115–124.
- [MDS22] Dheeraj Mekala, Chengyu Dong, and Jingbo Shang. *LOPS: Learning Order Inspired Pseudo-Label Selection for Weakly Supervised Text Classification*. 2022. DOI: [10.48550/ARXIV.2205.12528](https://doi.org/10.48550/ARXIV.2205.12528). URL: <https://arxiv.org/abs/2205.12528>.
- [MS20] Dheeraj Mekala and Jingbo Shang. “Contextualized weak supervision for text classification.” In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. 2020, pp. 323–333.
- [MZS20] Dheeraj Mekala, Xinyang Zhang, and Jingbo Shang. “META: Metadata-Empowered Weak Supervision for Text Classification.” In: *Conference on Empirical Methods in Natural Language Processing*. 2020.
- [Mik+13] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. “Efficient estimation of word representations in vector space.” In: *arXiv preprint arXiv:1301.3781* (2013).
- [Min+09] Mike Mintz, Steven Bills, Rion Snow, and Dan Jurafsky. “Distant Supervision for Relation Extraction without Labeled Data.” In: *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2 - Volume 2*. ACL '09.

- Suntec, Singapore: Association for Computational Linguistics, 2009, 1003–1011. ISBN: 9781932432466.
- [Mor+19] Ari Morcos, Haonan Yu, Michela Paganini, and Yuandong Tian. “One ticket to win them all: generalizing lottery ticket initializations across datasets and optimizers.” In: *Advances in neural information processing systems* 32 (2019).
- [MA20] Subhabrata Mukherjee and Ahmed Awadallah. “Uncertainty-aware self-training for few-shot text classification.” In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 21199–21212.
- [Nam+19] Jinseok Nam, Young-Bum Kim, Eneldo Loza Mencia, Sunghyun Park, Ruhi Sarikaya, and Johannes Fürnkranz. “Learning Context-dependent Label Permutations for Multi-label Classification.” In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, 2019, pp. 4733–4742. URL: <https://proceedings.mlr.press/v97/nam19a.html>.
- [NSo4] Hieu T Nguyen and Arnold Smeulders. “Active learning using pre-clustering.” In: *Proceedings of the twenty-first international conference on Machine learning*. 2004, p. 79.
- [NDH19] Vu-Linh Nguyen, Sébastien Destercke, and Eyke Hüllermeier. *Epistemic Uncertainty Sampling*. 2019. DOI: [10.48550/ARXIV.1909.00218](https://doi.org/10.48550/ARXIV.1909.00218). URL: <https://arxiv.org/abs/1909.00218>.
- [NSH22] Vu-Linh Nguyen, Mohammad Hossein Shaker, and Eyke Hüllermeier. “How to Measure Uncertainty in Uncertainty Sampling for Active Learning.” In: *Mach. Learn.* 111.1 (2022), 89–122. ISSN: 0885-6125. DOI: [10.1007/s10994-021-06003-9](https://doi.org/10.1007/s10994-021-06003-9). URL: <https://doi.org/10.1007/s10994-021-06003-9>.
- [NMC05] Alexandru Niculescu-Mizil and Rich Caruana. “Predicting good probabilities with supervised learning.” In: *Proceedings of the 22nd international conference on Machine learning*. 2005, pp. 625–632.
- [PSM14] Jeffrey Pennington, Richard Socher, and Christopher Manning. “GloVe: Global Vectors for Word Representation.” In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1532–1543. DOI: [10.3115/v1/D14-1162](https://doi.org/10.3115/v1/D14-1162). URL: <https://aclanthology.org/D14-1162>.
- [Pri04] Michael Prince. “Does active learning work? A review of the research.” In: *Journal of engineering education* 93.3 (2004), pp. 223–231.

- [Qia+18] Siyuan Qiao, Wei Shen, Zhishuai Zhang, Bo Wang, and Alan Yuille. “Deep co-training for semi-supervised image recognition.” In: *Proceedings of the european conference on computer vision (eccv)*. 2018, pp. 135–152.
- [RB21] Anant Raj and Francis Bach. *Convergence of Uncertainty Sampling for Active Learning*. 2021. DOI: [10.48550/ARXIV.2110.15784](https://doi.org/10.48550/ARXIV.2110.15784). URL: <https://arxiv.org/abs/2110.15784>.
- [RU11] Anand Rajaraman and Jeffrey David Ullman. “Data Mining.” In: *Mining of Massive Datasets*. Cambridge University Press, 2011, 1–17. DOI: [10.1017/CB09781139058452.002](https://doi.org/10.1017/CB09781139058452.002).
- [RPHo8] Jesse Read, Bernhard Pfahringer, and Geoff Holmes. “Multi-label Classification Using Ensembles of Pruned Sets.” In: *2008 Eighth IEEE International Conference on Data Mining*. 2008, pp. 995–1000. DOI: [10.1109/ICDM.2008.74](https://doi.org/10.1109/ICDM.2008.74).
- [Riz+21] Mamshad Nayeem Rizve, Kevin Duarte, Yogesh S Rawat, and Mubarak Shah. In *Defense of Pseudo-Labeling: An Uncertainty-Aware Pseudo-label Selection Framework for Semi-Supervised Learning*. 2021. DOI: [10.48550/ARXIV.2101.06329](https://doi.org/10.48550/ARXIV.2101.06329). URL: <https://arxiv.org/abs/2101.06329>.
- [Rot+18] Nadav Rotem et al. *Glow: Graph Lowering Compiler Techniques for Neural Networks*. 2018. DOI: [10.48550/ARXIV.1805.00907](https://doi.org/10.48550/ARXIV.1805.00907). URL: <https://arxiv.org/abs/1805.00907>.
- [SJT16] Mehdi Sajjadi, Mehran Javanmardi, and Tolga Tasdizen. “Regularization with stochastic transformations and perturbations for deep semi-supervised learning.” In: *Advances in neural information processing systems* 29 (2016).
- [San+19] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. *DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter*. 2019. DOI: [10.48550/ARXIV.1910.01108](https://doi.org/10.48550/ARXIV.1910.01108). URL: <https://arxiv.org/abs/1910.01108>.
- [Sch+14] Axel Schulz, Eneldo Loza Mencía, Thanh Tung Dang, and Benedikt Schmidt. “Evaluating multi-label classification of incident-related tweets.” In: *Proceedings of the Making Sense of Microposts (# Microposts 2014), Seoul, Korea (2014)*, pp. 7–11.
- [Scu65] H. Scudder. “Probability of error of some adaptive pattern-recognition machines.” In: *IEEE Transactions on Information Theory* 11.3 (1965), pp. 363–371. DOI: [10.1109/TIT.1965.1053799](https://doi.org/10.1109/TIT.1965.1053799).
- [STV11] Konstantinos Sechidis, Grigorios Tsoumakas, and Ioannis Vlahavas. “On the stratification of multi-label data.” In: *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2011, Athens, Greece, September 5-9, 2011, Proceedings, Part III* 22. Springer. 2011, pp. 145–158.

- [Sha48] C. E. Shannon. “A mathematical theory of communication.” In: *The Bell System Technical Journal* 27.3 (1948), pp. 379–423. DOI: [10.1002/j.1538-7305.1948.tb01338.x](https://doi.org/10.1002/j.1538-7305.1948.tb01338.x).
- [SGM22] Ruoqi Shen, Liyao Gao, and Yi-An Ma. *On Optimal Early Stopping: Over-informative versus Under-informative Parametrization*. 2022. DOI: [10.48550/ARXIV.2202.09885](https://doi.org/10.48550/ARXIV.2202.09885). URL: <https://arxiv.org/abs/2202.09885>.
- [Sri+14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting.” In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. URL: <http://jmlr.org/papers/v15/srivastava14a.html>.
- [SVL14] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. *Sequence to Sequence Learning with Neural Networks*. 2014. DOI: [10.48550/ARXIV.1409.3215](https://doi.org/10.48550/ARXIV.1409.3215). URL: <https://arxiv.org/abs/1409.3215>.
- [TRR22] Michael Tänzler, Sebastian Ruder, and Marek Rei. “Memorisation versus Generalisation in Pre-trained Language Models.” In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2022, pp. 7564–7578.
- [TGH15] Isaac Triguero, Salvador García, and Francisco Herrera. “Self-labeled techniques for semi-supervised learning: taxonomy, software and empirical study.” In: *Knowledge and Information systems* 42.2 (2015), pp. 245–284.
- [TV07] Grigorios Tsoumakas and Ioannis Vlahavas. “Random k-labelsets: An ensemble method for multilabel classification.” In: *Machine Learning: ECML 2007: 18th European Conference on Machine Learning, Warsaw, Poland, September 17-21, 2007. Proceedings 18*. Springer. 2007, pp. 406–417.
- [VEH20] Jesper E Van Engelen and Holger H Hoos. “A survey on semi-supervised learning.” In: *Machine Learning* 109.2 (2020), pp. 373–440.
- [Vas+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. *Attention Is All You Need*. 2017. DOI: [10.48550/ARXIV.1706.03762](https://doi.org/10.48550/ARXIV.1706.03762). URL: <https://arxiv.org/abs/1706.03762>.
- [Vig19] Jesse Vig. “A Multiscale Visualization of Attention in the Transformer Model.” In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. Florence, Italy: Association for Computational Linguistics, July 2019, pp. 37–42. DOI: [10.18653/v1/P19-3007](https://doi.org/10.18653/v1/P19-3007). URL: <https://www.aclweb.org/anthology/P19-3007>.

- [Wan+19] Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. *SuperGLUE: A Stickier Benchmark for General-Purpose Language Understanding Systems*. 2019. DOI: [10.48550/ARXIV.1905.00537](https://doi.org/10.48550/ARXIV.1905.00537). URL: <https://arxiv.org/abs/1905.00537>.
- [Wan+18] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. *GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding*. 2018. DOI: [10.48550/ARXIV.1804.07461](https://doi.org/10.48550/ARXIV.1804.07461). URL: <https://arxiv.org/abs/1804.07461>.
- [WWL20] Ziheng Wang, Jeremy Wohlwend, and Tao Lei. "Structured Pruning of Large Language Models." In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, 2020. DOI: [10.18653/v1/2020.emnlp-main.496](https://doi.org/10.18653/v1/2020.emnlp-main.496). URL: <https://doi.org/10.18653/v1/2020.emnlp-main.496>.
- [Wu+16] Yonghui Wu et al. *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation*. 2016. DOI: [10.48550/ARXIV.1609.08144](https://doi.org/10.48550/ARXIV.1609.08144). URL: <https://arxiv.org/abs/1609.08144>.
- [Xu+19] Jingjing Xu, Xu Sun, Zhiyuan Zhang, Guangxiang Zhao, and Junyang Lin. "Understanding and Improving Layer Normalization." In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett. Vol. 32. Curran Associates, Inc., 2019. URL: <https://proceedings.neurips.cc/paper/2019/file/2f4fe03d77724a7217006e5d16728874-Paper.pdf>.
- [Xun+20] Guangxu Xun, Kishlay Jha, Jianhui Sun, and Aidong Zhang. "Correlation networks for extreme multi-label text classification." In: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2020, pp. 1074–1082.
- [Yano01] Yiming Yang. "A study of thresholding strategies for text categorization." In: *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*. 2001, pp. 137–145.
- [Zaf+19] Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. "Q8BERT: Quantized 8Bit BERT." In: *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing - NeurIPS Edition (EMC2-NIPS)*. IEEE, 2019. DOI: [10.1109/emc2-nips53020.2019.00016](https://doi.org/10.1109/emc2-nips53020.2019.00016). URL: <https://doi.org/10.1109/emc2-nips53020.2019.00016>.

- [Zha+21] Yu Zhang, Zhihong Shen, Yuxiao Dong, Kuansan Wang, and Jiawei Han. *MATCH: Metadata-Aware Text Classification in A Large Hierarchy*. 2021. DOI: [10.48550/ARXIV.2102.07349](https://doi.org/10.48550/ARXIV.2102.07349). URL: <https://arxiv.org/abs/2102.07349>.
- [Zhe+21] Yi Zheng, Shixiang Tang, Guolong Teng, Yixiao Ge, Kaijian Liu, Jing Qin, Donglian Qi, and Dapeng Chen. "Online Pseudo Label Generation by Hierarchical Cluster Dynamics for Adaptive Person Re-Identification." In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 2021, pp. 8371–8381.
- [Zhu+22] Dawei Zhu, Michael A. Hedderich, Fangzhou Zhai, David Ifeoluwa Adelani, and Dietrich Klakow. *Is BERT Robust to Label Noise? A Study on Learning with Noisy Labels in Text Classification*. 2022. DOI: [10.48550/ARXIV.2204.09371](https://doi.org/10.48550/ARXIV.2204.09371). URL: <https://arxiv.org/abs/2204.09371>.

Part II

Appendix

FIGURES

| Rank | Name | Model |
|------|----------------------------|-----------------------|
| 1 | Microsoft Alexander v-team | Turing ULR v6 |
| 2 | JDExplore d-team | Vega v1 |
| 3 | Microsoft Alexander v-team | Turing NLR v5 |
| 4 | DIRL Team | DeBERTa + CLEVER |
| 5 | ERNIE Team - Baidu | ERNIE |
| 6 | AliceMind & DIRL | StructBERT + CLEVER |
| 7 | DeBERTa Team - Microsoft | DeBERTa / TuringNLRv4 |
| 8 | HFL iFLYTEK | MacALBERT + DKM |
| 9 | PING-AN Omni-Sinitic | ALBERT + DAAF + NAS |
| 10 | T5 Team - Google | T5 |

Figure A.1: GLUE Top 10 Models

A.1 PREDICTIVE

| Category | Binary Ensemble | Multi-Label | ML 8-Bit | Multi-Label | Multi-Label | Multi-Label |
|---------------------|-----------------|--------------|--------------|--------------|-------------|-------------|
| | ELECTRA | ELECTRA | ELECTRA | DistilBERT | LSTM | XGBoost |
| Macro | 0.869 | 0.867 | 0.862 | 0.853 | 0.813 | 0.842 |
| Micro | 0.863 | 0.860 | 0.859 | 0.850 | 0.803 | 0.837 |
| Face Mask General | 0.980 | 0.960 | 0.960 | 0.984 | 0.933 | 0.976 |
| Loudness | 0.938 | 0.931 | 0.937 | 0.915 | 0.870 | 0.928 |
| Luggage General | 0.893 | 0.921 | 0.908 | 0.897 | 0.847 | 0.877 |
| Passenger Rights | 0.817 | 0.847 | 0.838 | 0.802 | 0.764 | 0.773 |
| Punctuality | 0.837 | 0.846 | 0.853 | 0.852 | 0.785 | 0.841 |
| Seat Availability | 0.824 | 0.813 | 0.786 | 0.798 | 0.744 | 0.774 |
| Seating Comfort | 0.844 | 0.846 | 0.846 | 0.800 | 0.785 | 0.800 |
| Temperature General | 0.901 | 0.901 | 0.886 | 0.889 | 0.878 | 0.882 |
| Train Cancellation | 0.853 | 0.794 | 0.807 | 0.755 | 0.711 | 0.734 |
| Train Service | 0.700 | 0.701 | 0.694 | 0.743 | 0.677 | 0.707 |
| WLAN / Internet | 0.973 | 0.973 | 0.973 | 0.950 | 0.947 | 0.969 |

Table A.1: Comparison of the predictive performance of different model architectures on the holdout validation data via F1-Score

A.2 LATENCY

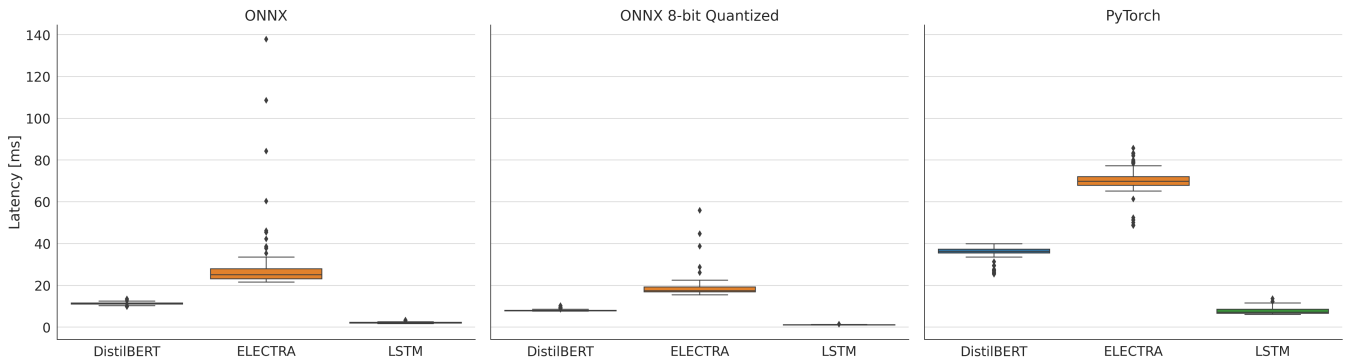


Figure A.2: Per-Sample Inference Latency for different model type with input sequence length 20

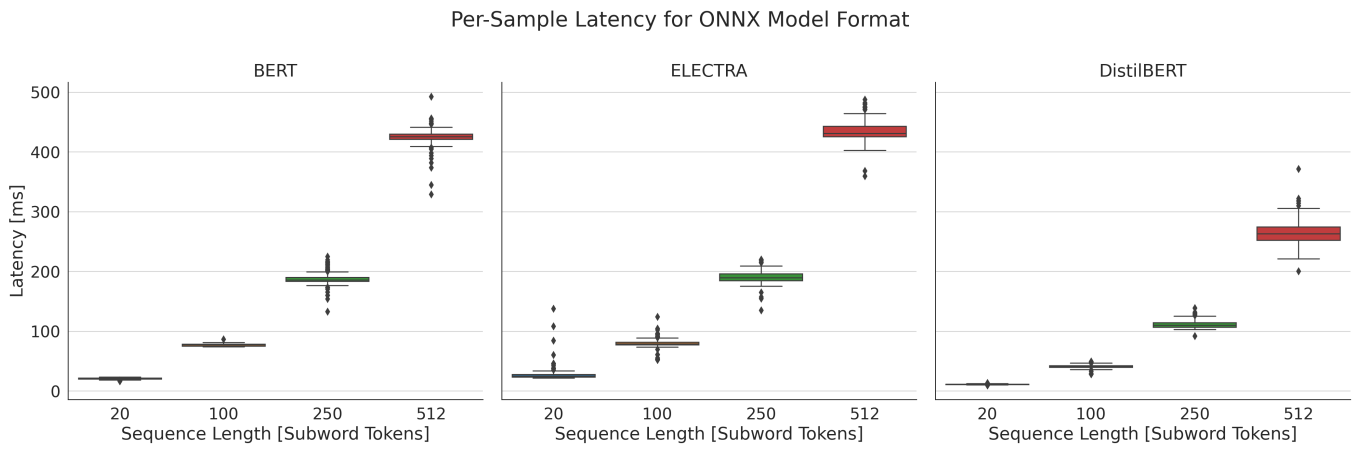


Figure A.3: Inference latency with ONNX runtime

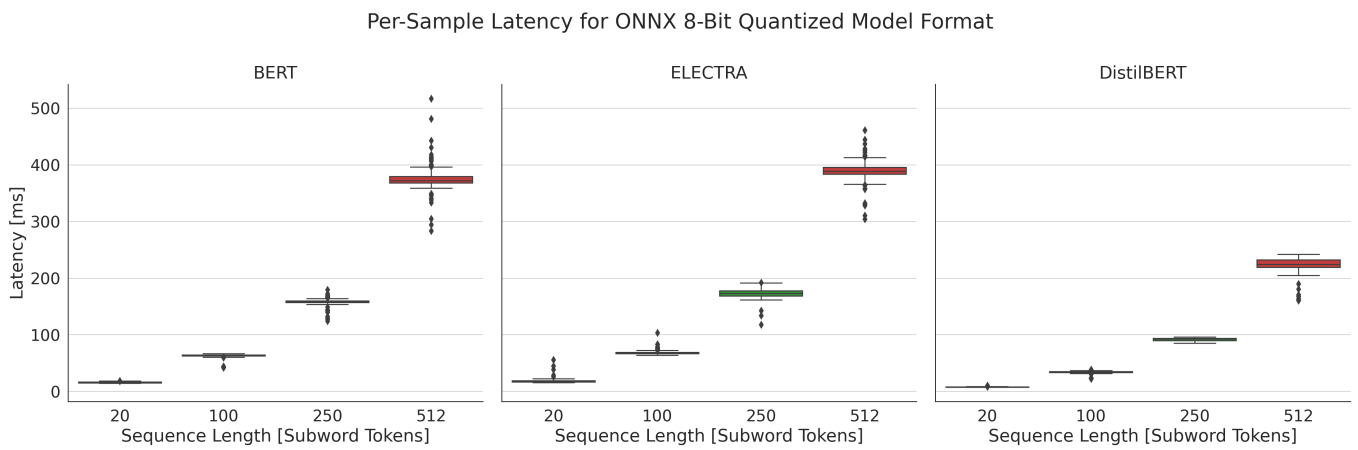


Figure A.4: Inference latency for 8-bit quantized model and ONNX runtime