ST. PÖLTEN UNIVERSITY
OF APPLIED SCIENCES

/Informatik & Security /fh/// st.pölten

Eine Evaluierung von LLMs für Mail-SPAM-Filterung

Das beste KI-Modell für einen lokalen Mail-SPAM-Filter ist?

Masterarbeit

zur Erlangung des akademischen Grades

Diplom-Ingenieur/in

eingereicht von

Ing. Ralph Holzer BSc

is221810

im Rahmen des
Studienganges Information Security an der Fachhochschule St. Pölten

Betreuung
Betreuer/in: FH-Prof. Dipl.-Ing. Dr. Martin Pirker, Bakk.
Mitwirkung:

Ehrenwörtliche Erklärung

Titel: Eine Evaluierung von LLMs für Mail-SPAM-Filterung

Art der Arbeit: Masterarbeit

Autor: Ing. Ralph Holzer BSc

Matrikelnummer: is221810

Ich versichere, dass

- ich diese Arbeit selbständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich sonst keiner unerlaubten Hilfe bedient habe.
- ich das Thema dieser Arbeit bisher weder im Inland noch im Ausland einem Begutachter/einer Begutachterin zur Beurteilung oder in irgendeiner Form als Prüfungsarbeit vorgelegt habe.
- diese Arbeit mit der vom Begutachter/von der Begutachterin beurteilten Arbeit übereinstimmt.

Ich erkläre hiermit, dass

☒ ich ein Large Language Model (LLM) verwendet habe, um die Arbeit Korrektur zu lesen.

☐ ich ein Large Language Model (LLM) verwendet habe, um Teile des Inhalts der Arbeit zu erstellen. Ich versichere, dass ich jeden generierten Satz/Absatz mit der Originalquelle zitiert habe. Das genutzte LLM ist an entsprechenden Stellen durch eine Fußnote ausgewiesen.

☐ im Zuge dieser Arbeit kein Large Language Model (LLM) zum Einsatz gekommen ist.

Ort, Datum *Unterschrift*

Kurzfassung

Im heutigen Internet sind Mails nach wie vor eine zentrale Anwendung. Leider wird dieser universelle Kommunikationsdienst immer wieder von unerwünschten Mails gestört. Diese unerwünschten Mails, von Schadcode über Phishing bis zu SPAM-Mails, sind im besten Fall nur nervig, aber sie können auch Schaden anrichten. All diese Arten von Mails haben gemeinsam, dass sie aus Text bestehen.
Damit können sie von einem SPAM-Filter, der auf den Text fokussiert ist, erkannt und aussortiert werden. SPAM-Filter sind keine neue Lösung, sie müssen aber ständig angepasst werden, da auch der SPAM sich ständig ändert. Es ist ein andauernder Wettlauf zwischen den SPAM-VersendernInnen und den VerteidigernInnen, den Serveradministratoren.

Dieser jahrzehntelange Konflikt wird nun durch den Aufstieg von Technologien der künstlichen Intelligenz (KI) beeinflusst, wobei beide Seiten gewinnen oder verlieren können. Auch ein bereits auf KI basierender SPAM-Filter liefert jeden Monat schlechtere SPAM-Erkennungsraten, wenn er nicht aktiv angepasst wird. Das Ziel dieser Arbeit ist daher, einen bereits bestehenden KI-SPAM-Filter durch den Einsatz von großen Sprachmodellen (LLMs) zu verbessern.

Der bestehende Filter arbeitet mit 6 klassischen neuronalen Netzwerken (NN) :

Dense, LSTM, Bi-LSTM, Convolutional und 2x Multi-Convolutional;

Dabei werden die Mails je nach Länge und Sprache in 4 x 4 Gruppen aufgeteilt. Daraus ergeben sich 16 verschiedene Modelle und Wörterbücher. Durch diese Aufteilung werden jedes einzelne Wörterbuch und das NN klein und die Berechnung wird effizient gehalten. Allerdings ist das Training aufwendig und die Sprachunterscheidung und Längenaufteilung machen den Filter auch komplex.

Aktuelle Fortschritte im KI-Bereich legen nahe, dass ein LLM diesen Filter ersetzen kann, da diese schon mit vielen Daten vortrainiert sind und verschiedenste Sprachen ausgezeichnet beherrschen.
Folgende LLMs werden getestet:

Bart, bge-reranker, Llama-3.2 1B/3B, Phi-3/4, Teuken, Mistral/7B/8x7B/Nemo, SmolLM2, DeepSeek: R1 und GPT-2/3.5;

Bei der Gelegenheit werden auch neue Ideen für die Vorverarbeitung betrachtet, um die Qualität der Daten und damit die Genauigkeit der SPAM-Klassifizierung zu erhöhen:

- Auswahl der Trainingsdaten über den Ähnlichkeits-Hash Nilsimsa
- Übersetzen der Mails auf Englisch
- Entfernung oder Ersetzen von Mailadressen, URLs, Zahlen, Satzzeichen und Stoppwörtern
- Text auf den Wortstamm zurückführen
- Text mit Zusatzinformationen anreichern: grammatikalische Stellung und Wichtigkeit der Wörter
- Die Kontextlänge wird variiert
- Die Wörterbuchgröße wird variiert

Der Energieverbrauch von CPU oder GPU wird genauer betrachtet und verglichen. Die ethischen Aspekte eines LLMs werden ebenfalls kurz betrachtet.

Die NN wurden in einer lokalen Einrichtung vor Ort mit bis zu 2,2 Millionen Mails trainiert und getestet.
Nilsimsa hat sich für die Auswahl der Trainingsdaten und für die Reduktion der Datenmenge bewährt.
Für die klassischen NN konnten durch Anpassung folgender Parameter die Ergebnisse verbessert werden: Wörterbuchgröße und Kontextlänge.
Das Übersetzen der Mails ins Englische hat bei fast allen LLMs die Ergebnisse um bis zu 2% verbessert. Bei den klassischen NN hatte das Übersetzen keinen nennenswerten Einfluss auf die Ergebnisse. Entgegen der Erwartung lieferte keines der getesteten LLMs eine bessere SPAM-Erkennungsrate (max. 87,6%) als die bestehenden klassischen NN (max. 97,5%).

Die meisten getesteten Vorverarbeitungsschritte haben nur bei einfachen Dense- und LSTM-NN geholfen und das auch nur in Ausnahmefällen wie bei kleinen Trainingsdatensätzen. Bei komplexeren neueren NN (Bi-LSTM, Convolutional, Multi-Convolutional und LLMs) gab es dadurch keine Verbesserung.
Der Energieverbrauch ist nicht so groß wie erwartet. Es gibt deutliche Unterschiede zwischen den klassischen NN und den LLMs. Die LLMs verbrauchten erhebliche Ressourcen, ohne die SPAM-Filterung zu verbessern.
Man kann durch einfache Anpassung viel Rechenleistung und Zeit sparen. Bei der Vorverarbeitung mit Anwendung des Filters sollte man die CPU verwenden. Fürs Training und Übersetzen von Mails die GPU.

Abstract

In today's Internet, e-mail is still a central application. Unfortunately, this universal communication service is repeatedly disrupted by unwanted mail. These unwanted mails, from malicious code to phishing to SPAM mails, are at best only annoying, but they can also cause damage. All these types of mail have in common that they consist of text.

This allows them to be recognized and sorted out by a SPAM filter that focuses on the text. SPAM filters are not a new solution, but they have to be constantly adapted, as SPAM is also frequently changing, and it is a constant race between the SPAM senders and the defenders, the server administrators.

This decades-long conflict is now influenced by the rise of artificial intelligence (AI) technologies, where both sides can win or lose. Even a SPAM filter already based on AI delivers worse SPAM detection rates every month if it is not actively adapted. The aim of this thesis is therefore to improve an already existing AI SPAM filter by using large language models (LLMs).

The existing filter works with 6 classic neural networks (NN):

Dense, LSTM, Bi-LSTM, Convolutional and 2x Multi-Convolutional;

It divides the emails into 4 x 4 groups depending on their length and language. This results in 16 different models and dictionaries. This division means that each individual dictionary and the NN are small, and the calculation is kept efficient. However, the training is time-consuming and the language differentiation and length division also make the filter complex.

Current advances in AI suggest that an LLM can replace this filter, as they are already pre-trained with a lot of data and have an excellent understanding of a wide variety of languages. The following LLMs are being tested:

Bart, bge-reranker, Llama-3.2 1B/3B, Phi-3/4, Teuken, Mistral/7B/8x7B/Nemo, SmolLM2, DeepSeek R1 and GPT-2/3.5;

New ideas for preprocessing will also be considered to increase the quality of the data and thus the accurancy

of the SPAM classification:

- Selection of training data via the similarity hash Nilsimsa
- Translation of emails into English
- Removal or replacement of email addresses, URLs, numbers, punctuation marks and stop words
- Replace words with their word stem
- Enrich text with additional information: grammatical position and importance of words
- The context length is varied
- The dictionary size is varied

The energy consumption of the CPU or GPU is examined and compared in more detail. The ethical aspects of an LLM are also briefly considered.
The NNs were trained and evaluated in a local facility on-site with up to 2.2 million mails.
Nilsimsa proved its worth for the selection of training data and for reducing the amount of data.
For the classic NNs, the results could be improved by adjusting the following parameters: dictionary size and context length.

Most of the tested preprocessing steps only helped with simple Dense and LSTM NNs, and only in exceptional cases, such as with small training data sets. For more complex, newer NNs (Bi-LSTM, Convolutional, Multi-Convolutional and LLMs), there was no improvement.
Translating the emails into English improved the results for almost all LLMs by up to 2%. For the classic NN, the translation had no significant influence on the results.
Contrary to expectations, none of the LLMs tested delivered a better SPAM detection rate (max. 87.6%) than the existing classic NNs (max. 97.5%).

The energy consumption is not as high as expected. There are clear differences between the classic NNs and the LLMs. The LLMs consumed significantly more resources without improving SPAM filtering.
But a lot of computing power and time can be saved by simple adaptation. The CPU should be used for the preprocessing application of the filter. Use the GPU for training and translating mails.

Inhaltsverzeichnis

1 Einleitung . . . **1**

1.1 Problemstellung . . . 2

1.1.1 Warum einen eigenen lokalen SPAM-Filter verwenden? . . . 2

1.2 Lösung . . . 3

1.2.1 Fokus . . . 4

1.2.2 Nicht im Fokus . . . 4

1.3 Struktur der Arbeit . . . 4

2 Grundlagen . . . **7**

2.1 Mailaufbau . . . 7

2.1.1 Mail-Header . . . 7

2.1.2 Mail-Body . . . 9

2.2 SPAM-Filter allgemein . . . 11

2.2.1 Anhänge und Web-Links . . . 12

2.2.2 Vorverarbeitung . . . 13

2.2.3 Übersetzung . . . 15

2.2.4 KI . . . 16

2.3 SPAM-Filter mit klassische NN . . . 19

2.3.1 SPAM-HAM-Verhältnis von 1:1 . . . 20

2.4 Verwendete klassische NN des bestehenden alten SPAM-Filters . . . 20

2.4.1 Dense . . . 20

2.4.2 LSTM . . . 21

2.4.3 Convolutional NN . . . 21

2.4.4 Zero Shoot Classification . . . 22

2.5 Entwicklungen im KI-Bereich . . . 22

2.5.1 Transformer 22
2.5.2 BERT 22
2.5.3 ChatGPT 23
2.5.4 LLaMA 23
2.5.5 Open Weights 24
2.5.6 Absichtliche Grenzen 24
2.5.7 Knowledge-Cut-off-Date 24
2.5.8 Ende des Wachstums 25
2.5.9 Quantisierung 25
2.5.10 Mixtral - MoE 26
2.5.11 Effizienz 26
2.5.12 LLaMA-Factory 26
2.6 SPAM-Filter mit LLM 27
2.6.1 Vergleich der Ergebnisse 27
2.7 Angriffe auf KI 28
2.8 Hardware und Container 28
2.8.1 Prozessor 28
2.8.2 Video-Speicher: VRAM 28
2.8.3 Podman 29
2.9 Bedarf an elektrischer Energie 29
2.10 Rechtliche Aspekte 30

3 Stand der Forschung 31
3.1 Kombinierte SPAM-Filter 31
3.2 SPAM-Filter mit NN 31
3.2.1 SPAM-Filter mit BERT 33
3.2.2 SPAM-Filter mit neueren LLMs 33
3.3 Angriffe auf KI 34

4 Methodik 35
4.1 Ausgangslage 35
4.2 Aufbau des bestehenden SPAM-Filtersystems 36
4.2.1 Mailgateway 36

4.2.2 Anti-SPAM-Server 36
4.2.3 Contentchecker 37
4.2.4 Mailarchivserver 37
4.2.5 Mailbox-Server 38
4.2.6 SPAM-Statistik 38
4.3 Daten-Beschaffung 40
4.3.1 Testdaten 40
4.3.2 Datensatz Nr.1 emails1.csv 41
4.3.3 Mails vom Mailarchiv 41
4.3.4 Datensatz Nr.2 mail65.csv 41
4.3.5 Datensatz Nr.3 mail65plus.csv 42
4.3.6 Datensatz Nr.4 1Monat18042024.csv 42
4.3.7 Datensatz Nr.5 1monU65.csv 42
4.3.8 Datensatz Nr.6 6jSpamHam1q24.csv 42
4.4 Neuronale Netzwerke (NN) 43
4.4.1 Offene Fragen 44
4.4.2 Testablauf 44
4.5 Verwendete Software 47
4.5.1 Python 47
4.5.2 Jupyter 48
4.5.3 PyTorch 48
4.5.4 Argos-Translate 48
4.5.5 spaCy 49
4.5.6 Nvidia CUDA 49
4.5.7 Nilsimsa 49
4.6 Aufbau der Testumgebung 50
4.7 Containerarten 51
4.8 Hardware 51
4.9 Vorverarbeitung 52
4.9.1 Text reinigen 53
4.9.2 Textvorbereitung 53
4.9.3 Übersetzung 56

4.10 Test der Modelle . . . 57
4.10.1 Auswahl der Modelle . . . 57
4.10.2 LLM-Chat-Bot . . . 57
4.10.3 LlaMA-Factory . . . 58
4.10.4 Änderungen an den klassischen Modellen . . . 58
4.10.5 BERT und LLM . . . 58
4.11 Bedarf an elektrischer Energie . . . 59
4.12 Rechtliche Aspekte . . . 59

5 Herangehensweise . . . 63
5.1 Szenario . . . 63
5.2 Testablauf . . . 63
5.3 Vorverarbeitung . . . 63
5.3.1 Übersetzung . . . 66
5.3.2 FastText . . . 66
5.3.3 Argos-Translate . . . 66
5.3.4 Übersetzung anpassen . . . 67
5.3.5 Nilsimsa berechnen . . . 67
5.3.6 Nilsimsa Grenzwert . . . 68
5.4 Test der Modelle . . . 68
5.4.1 Auswahl der Modelle . . . 68
5.4.2 LLM-Chat-Bot . . . 70
5.4.3 LLaMA-Factory . . . 70
5.4.4 Änderungen an den klassischen Modellen . . . 72
5.4.5 BERT LLM . . . 73
5.4.6 Gleitkommazahl . . . 73
5.5 Bedarf an elektrischer Energie . . . 73
5.5.1 CPU oder GPU . . . 74
5.6 Rechtliche Aspekte . . . 74
5.6.1 DSGVO . . . 74
5.6.2 AI-Act . . . 74

6 Ergebnisse . . . 75

6.1 Beantwortete Fragen 75
6.2 Vorverarbeitung 75
6.2.1 Textvorbereitung 75
6.2.2 Argos-Translate 80
6.2.3 Auswirkungen der automatischen Übersetzung 80
6.3 Test der Modelle 82
6.3.1 LLM-Chat-Bot 82
6.3.2 LlaMA-Factory 83
6.3.3 Änderungen an den klassischen Modellen 84
6.3.4 Gleitkommazahl 86
6.3.5 Vergleich der Modelle 86
6.4 Angriffe auf KI 87
6.5 Bedarf an elektrischer Energie 88
6.5.1 Vorverarbeitung 88
6.5.2 Übersetzung 89
6.5.3 Klassische Modelle trainieren 89
6.5.4 Lokale LLMs anpassen/trainieren 89
6.5.5 Aufwand für das grundlegende Training eines LLMs am z.B. Teuken-7B 90
6.6 Rechtliche Aspekte 91
6.6.1 DSGVO 92
6.6.2 AI-Act 92

7 Diskussion 101
7.1 Diskussion Textanpassung 101
7.2 Diskussion Stemmer 101
7.3 Diskussion LPD 101
7.4 Diskussion Genauigkeit 101
7.5 Diskussion Energieverbrauch CPU oder GPU 102
7.6 Diskussion Energieverbrauch Klassifizierung 102
7.7 Diskussion Energieverbrauch Training 103

8 Schlussfolgerung 105
8.1 Lessons Learned 107

8.1.1 GPU-Verwaltung . . . 107
8.1.2 Arbeiten mit mehreren Grafikkarten . . . 107
8.1.3 Mythen . . . 107
8.2 Weiterführende Arbeiten . . . 108
8.2.1 Daten . . . 108
8.2.2 Neuronales Netzwerk . . . 109

Abbildungsverzeichnis . . . **110**

Tabellenverzeichnis . . . **113**

Auflistung von Quelltext und Abhängigkeiten . . . **114**

Akronyme . . . **117**

Literatur . . . **121**

A Anhang Jupyter-Notebooks und Skripte . . . **141**
A.1 Archivierung, Umwandlung in CSV . . . 141
A.1.1 CheckAntispam.py . . . 141
A.1.2 mail2rmailSkript.py . . . 156
A.1.3 rmailbox_csv_read.py . . . 158
A.2 Längen- und Spracherkennung . . . 161
A.2.1 labelCSVmail.ipynb . . . 161
A.3 Nilsimsa . . . 163
A.3.1 nilLab3.ipynb . . . 163
A.3.2 nilLabChoose.ipynb . . . 166
A.4 Verhältnis anpassen . . . 170
A.4.1 nilLabDiffnil-2x.ipynb . . . 170
A.5 Stemmer, Lemma, LPD, und Übersetzen . . . 172
A.5.1 replaceNER.ipynb . . . 172
A.5.2 removeNER.ipynb . . . 175
A.5.3 antispamSpacy.ipynb . . . 178
A.5.4 labelCSVmailArgosTranslate.ipynb . . . 196
A.6 Training und Testsklassische NN . . . 205

A.6.1 antispam.ipynb . . . 205
A.6.2 compare.ipynb . . . 223
A.7 Training und Tests LLM . . . 233
A.7.1 llm-classifier.ipynb . . . 233
A.7.2 csv2jsonAlpaca3.ipynb . . . 244
A.8 NN Auswahl . . . 246
A.8.1 cpKiMpodeli.py . . . 246
A.8.2 bestModeledit.ipynb . . . 247

B Anhang Python-Pakete . . . 249
B.1 laptop.pip . . . 249
B.2 notebook.pip . . . 252
B.3 desktop.pip . . . 259
B.4 translate.pip . . . 263
B.5 rtx2070.pip . . . 266
B.6 llm.pip . . . 271

C Anhang Containerfiles . . . 277
C.1 Containerfile.jupyter-notebook . . . 277
C.2 Containerfile.translate . . . 281
C.3 Containerfile.llm . . . 284
C.4 Dockerfile.LLaMA-Factory . . . 286

1. Einleitung

Bei der Nachrichtenübermittlung im heutigen Internet ist die traditionelle Mail immer noch eine Kernanwendung. Die Mail bietet einen niedrigschwelligen Zugang zur universellen Kommunikation zwischen Internet-TeilnehmernInnen. Eines der Hauptprobleme ist der Missbrauch in Form von SPAM, der massenhaften Versendung von Werbung für Waren und Dienstleistungen von zweifelhaftem Nutzen oder schlichtweg Betrug. Die Zahl der Mail-Identitäten, die leicht erstellt werden können, ist unbegrenzt, und jeder kann an jeden Nachrichten senden. Es gibt viele Lösungen zur Bekämpfung von SPAM, und jeder Mailserver muss heute eine Art von SPAM-Filter beinhalten. Es ist ein ständiger technologischer Wettlauf zwischen SPAM-VersenderInnen und Mailserver-Administratoren.

Die Einrichtung und laufende Wartung eines Mail-Servers sind aufwändig. Er erfordert neben der Logistik für die Verarbeitung von Mails auch einen erheblichen Aufwand, um ihn vor Missbrauch und SPAM zu schützen - aus der Sicht der Übertragung (z.B. Greylisting, Blocklisting, Ratenbegrenzung, Reputationsmanagement für Server-IP-Adressen), der Absenderüberprüfung (z.B. SPF, DKIM, DMARC), der inhaltsbasierten Überwachung (z.B. Textanalyse, eingebettete Web-Links, Überprüfung von Anhängen) und mehr, um Mails auf unerwünschten SPAM zu filtern. Während die Abwehrseite alle eingehenden Mails analysiert, um echte Mails von SPAM-Kampagnen zu unterscheiden, die auf ein bösartiges Ziel hinarbeiten, setzen SPAM-VersenderInnen neue Techniken ein, um jede gesendete Mail zu randomisieren, zu maskieren und effektiv zu individualisieren, damit sie schwerer entdeckt und blockiert werden kann.

Heutzutage verlassen sich viele Endnutzer auf die Dienste der großen globalen (kostenlosen) Mail-Anbieter (z.B. GMail, GMX, Outlook, Yahoo, ...). Sie befassen sich nur wenig mit dem Problem von SPAM und nehmen in Kauf, dass gelegentlich eine Mail verloren geht oder SPAM in ihrem Posteingang landet. Für Unternehmen ist die Situation schwieriger: Es liegt in ihrem eigenen Interesse, aus Gründen des Datenschutzes oder aufgrund gesetzlicher Vorschriften einen eigenen Mailserver zu betreiben. Daher sind sie gezwungen, das SPAM-Problem in den Griff zu bekommen.

Aus Gesprächen mit der Administration eines echten geschäftlichen Mailservers, der etwa 1000 Benutzer bedient, mit etwa 2,2 Millionen empfangenen Mails pro Jahr, mit Mails in verschiedenen Sprachen (siehe auch Tabelle 4.6), haben wir erfahren, dass das Greylisting zwar immer noch wirksam ist, die Menge an SPAM aber weiter zunimmt - die Situation muss verbessert werden. Da SPAM-VersenderInnen nun LLMs verwenden, um individualisierten SPAM [1] zu zu erzeugen, ist es dringend notwendig, Möglichkeiten zu erforschen, LLMs auch zur Erkennung dieser neuen, schwierigen Art von SPAM einzusetzen.

1.1. Problemstellung

Ein wichtiges Merkmal von Mails war früher der Absender-Server, aber heutzutage haben viele Unternehmen ihre lokalen Mail-Server zu gemeinsamen Cloud-Hostern verlagert. Es gibt keinen klaren Unterschied mehr zwischen einem beliebigen, in der Cloud gehosteten Mailserver und dem Cloud-Mailserver eines etablierten Unternehmens. Darüber hinaus werden auch die Passwörter von Mailkonten gehackt, sogar die Mailkonten von Administratoren, so dass der Absender keine zuverlässige Information mehr darstellt. Insgesamt ist das Einzige, was ausgewertet werden kann und bestimmt, ob eine Mail legitim oder SPAM ist, zunehmend der Inhalt der Mail, der Text selbst.

Jede SPAM-Filter-Lösung braucht eine periodische Anpassung der Strategie. Daher werden bei der täglichen manuellen SPAM-Bekämpfung jene SPAM-Mails, die den bestehenden SPAM-Filter umgangen haben, erst analysiert, um den SPAM-Filter anzupassen. Hier 2 Beispiele:

- Neue Texte aus bekannten SPAM-Mails werden zur Text-Filter-Liste hinzugefügt.
- Ein neues Dateiformat, das der Virenscanner ignoriert. Diese Dateiendung wird in die Blockadekategorie des SPAM-Filters aufgenommen.

In beiden Fällen werden manuell Regeln erstellt, damit dieser SPAM in Zukunft erkannt und blockiert wird. Diese Arbeit erfolgt manuell und kostet Zeit. Es ist naheliegend, eine Lösung mit Machine Learning (ML, siehe Abschnitt 3.2) zu erstellen, da dieser die Regeln für den SPAM-Filter automatisch erstellen kann.

1.1.1. Warum einen eigenen lokalen SPAM-Filter verwenden?

In unserer Recherche zu dem Thema zeigte sich, dass viele SPAM-Filter zwar gute Ergebnisse liefern, allerdings meist nur für englischsprachige Mails. Bei anderen Sprachen fällt die Erkennungsrate stark ab [2]. Allgemeine SPAM-Filter werden meist nur mit dem SPAM trainiert, der ist weltweit ja relativ gleich, die echten Mails eines Unternehmens, der HAM (erwünschte Mails), sind aber von Unternehmen zu Unter-

nehmen stark unterschiedlich. Auch wird der SPAM immer gezielter, siehe Spear-Phishing [3] und CEO-Fraud [4]. Daher macht es Sinn, einen SPAM-Filter mit den eigenen Daten zu trainieren, da dadurch der HAM-Teil besser erkannt wird.

Die meisten Papers zum Thema SPAM-Filterung mit ML verwenden nur sehr kleine Mail-Datensätze, oft nur eine vierstellige Mailanzahl mit fast nur englischsprachige Mails, siehe die Ergebnisse in [5], [6] und [7]. Diese Arbeiten berichten über schnelle Erfolge und sehr gute Ergebnisse. In der Realität sind diese Ergebnisse aber unerreichbar, da aktueller SPAM inzwischen ganz anders aufgebaut ist. Die Details zu einem dieser Datensätze, dem Datensatz 1, sind hier zu finden: Abschnitt 4.3.2. Ergebnisse dazu sind in der Tabelle 6.6x). Er liefert immer sehr gute Ergebnisse, leider nur für englischsprachige Mails. Zugegeben, die englischsprachigen Mails sind im globalen Internet am weitesten verbreitet, regional betrachtet sieht das aber meist anders aus, für ein aktuelles Bsp. siehe Tabelle 4.6.

Auch wird der SPAM-Filter meist nur in der Theorie umgesetzt, der Autor arbeitet aber schon seit drei Jahren mit einem KI-SPAM-Filter im Produktionsnetzwerk eines Unternehmens, das pro Jahr rund 2,2 Millionen Mails empfängt. Da es sich um ein österreichisches Unternehmen handelt, sind über 70% der Mails deutsch.

1.2. Lösung

Ein bestehender KI-Mail-SPAM-Filter, wie er von einem Unternehmen bereitgestellt wird, wird auf mögliche Verbesserungen hin untersucht. Der in dieser Bachelorarbeit [8] entworfenen KI-SPAM-Filter, der auf klassischen künstlichen neuronalen Netzwerken (NN) basiert, wurde vor 3 Jahren in die bestehende SPAM-Klassifizierung integriert. Der klassische NN-SPAM-Filter arbeitet mit folgenden klassischen neuronalen Netzwerken (NN):

Dense, LSTM, Bi-LSTM, Convolutional und Multi-Convolutional.

Genauer gesagt wird in dieser Arbeit die Verwendung von Large Language Models (LLMs) für das Problem der SPAM-Filterung evaluiert. Aktuellste LLMs können ja den Kontext von Text richtig einordnen, unabhängig von der Form, der Sprache oder der Anzahl der Rechtschreibfehler. Daher ist davon auszugehen, dass sie einem auch bei der SPAM-Klassifizierung behilflich sein können. Die folgenden LLMs werden mit einem realen Produktions-Mail-Datensatz getestet:

Bart, bge-reranker, LLaMA-3.2 1B/3B, Phi-1.5/3/4, Teuken, Mistral 7B/8 x 7B/Nemo, SmolLM2, DeepSeek R1 und GPT 2/3.5.

Weiters werden auch wichtige änderbare Parameter des NN angepasst und auch hilfreiche Ideen auf Brauch-

barkeit untersucht:

- Wörterbuchgröße und Kontextlänge variieren
- Auswahl der unterschiedlichsten Mails für das Training
- Stemmer und Lemma-Algorithmus fürs Vereinfachen der Daten
- Genauere Analyse der Wortbedeutung im Satz mit POS und DEP
- Entfernen oder Ersetzen von Daten wie: URLs, Mail-Adressen und Zahlen
- Ist eine CPU oder eine GPU für die Aufgabe besser geeignet?
- Übersetzung der Mails ins Englische
- Wie viel Rechenzeit und Energie brauchen die einzelnen Aufgaben und Modelle?
- Wie sieht es mit den rechtlichen Aspekten des KI-SPAM-Filters aus?

1.2.1. Fokus

Der Text der Mail, also der Teil, den der Mailclient direkt anzeigt. Er besteht aus natürlicher Sprache (NLP). Für den Rest der Mails gibt es bereits sehr gute SPAM-Filtersysteme.
Wenn es sich um eine HTML-Mail handelt, wird sie in Text umgewandelt. Es wird nur der aktuellste Teil der Mail betrachtet, keine ältere Kommunikation, die oft auch angehängt ist. Die ältere Kommunikation stammt normalerweise von einem anderen Autor, sie wird oft von SPAM-VersenderInnen zur Täuschung [9] verwendet. Auch Mails, die auf Schadcode hinter Web-Links verweisen oder Schadcode im Anhang haben, bestehen aus bestimmten Texten, über die sie identifiziert werden können. Daher könnten auch Schadcode-Mails oft alleine über den Text erkannt und als SPAM eingestuft werden. Auch bei einer sehr guten Erkennungsrate sollte trotzdem zusätzlich noch ein traditioneller Virenscanner in Gebrauch sein.

1.2.2. Nicht im Fokus

Anhänge und Metadaten werden bei dieser Lösung nicht betrachtet, dafür gibt es schon gute Lösungen (z.B. Greylisting, Blockliste und Sandbox, siehe Abschnitt 2.1.1). Mails mit einem Terminanhang und Mails im alten Outlook-RTF-Format (Rich Text Format ist eine proprietäre Lösung von Microsoft aus 1987) werden ignoriert, da dafür keine einfache Umwandlung in Text verfügbar ist.

1.3. Struktur der Arbeit

In Kapitel 1 werden kurz die Aufgabe, die gelösten Probleme und die Motivation dahinter erklärt. Kapitel 2 gibt zunächst einen Überblick über die SPAM-Filterung und die Entwicklungen bei LLMs. In Kapitel 3

wird auf den Stand der Forschung für SPAM-Filter und neuronale Netzwerke eingegangen.
Dann beschreibt Kapitel 5 die Situation der Mail-Filterung im Detail, von der bestehenden Einrichtung über die Verarbeitungs-/Filterungsstufen bis hin zu unseren geplanten Tests, die mit NN durchgeführt werden. In Kapitel 4 sind die gefundenen Lösungen und die getroffenen Entscheidungen beschrieben. Kapitel 6 präsentiert die Ergebnisse, die wir in unseren Experimenten erhalten haben. Kapitel **??** bietet zusätzliche Diskussionen und Überlegungen.
Die Arbeit schließt mit der Zusammenfassung der Ergebnisse Kapitel 8 ab, bevor noch auf mögliche zukünftige Projekte eingegangen wird. Am Ende im Kapitel A sind Jupyter-Notebooks und Skripte, im Kapitel B gibt es die genauen Python-Paket-Versionen und die Containerfiles sind im Kapitel C zu finden.

2. Grundlagen

Viele Teile der ersten beiden Abschnitte dieses Kapitels wurden von [8] übernommen, da die Grundlagen gleich geblieben sind.

Im weiteren Text wird das Wort SPAM für alle Arten von unerwünschten Mails stehen, unabhängig davon, ob Schadcode, Phishing oder ein unerwünschter Newsletter, denn die Anforderung des Empfängers ist immer dieselbe. Er will solche Nachrichten blockiert haben. Zwar sind Schadcode- oder Phishing-Mails ungleich gefährlicher als ein nerviger Newsletter, aber auch diese Art von Mails besteht aus Text, der von einem SPAM-Filter erkannt wird.

SPAM-Mails sind kein neues Phänomen. Das erste SPAM-Mail wurde bereits 1978 versendet, es war eine unbestellte Werbung für einen neuen Computer. Seitdem gibt es immer ein technisches Wettrüsten zwischen den SPAM-VersenderInnen und den SPAM-Filtern, die diese automatisch erkennen und aussortieren sollten [10].

Leider wird jede technische Neuerung, die das Erkennen von SPAM leichter macht, auch immer schnell von den SPAM-VersenderInnen verwendet. Die jüngsten Fortschritte in der Künstlichen Intelligenz (KI) sind hier leider keine Ausnahme und zwingen jeden Mail-Provider sich ebenfalls damit zu beschäftigen, um weiterhin das Medium Mail vor Überflutung durch SPAM zu schützen. Da dadurch alte Anti-SPAM-Lösungen immer weniger wirksam sind.

2.1. Mailaufbau

Eine Mail gliedert sich in zwei Teile: in den Header und den Body. Der Body ist der Text der Mail, der im Mailclient angezeigt wird, inklusive Betreff, Anhänge, Empfänger und auch die Mailadressen.

2.1.1. Mail-Header

Das Mail selbst beginnt mit dem Mailheader, wo alle Metadaten zu finden sind. Auch die Adresse des/-der AbsenderIn kann von der im Body angezeigten abweichen! Hier finden sich der verwendete Mailclient

und die Übertragungscodierung. Der größte Teil des Headers besteht aus Metainformationen die am Übertragungsweg gesammelt werden. Wie bei einem Poststempel auf einem Briefkuvert hinterlässt jeder Mailserver, welcher die Mail übertragen hat, Spuren. Es macht daher durchaus Sinn, sich diese Daten genauer anzuschauen [11]. Üblicherweise wird dafür eine Kombination aus mehreren Verfahren verwendet [12].

Protokoll-Filter

Ein Protokollfilter überprüft, ob das SMTP-Protokoll eingehalten wird. SPAM-VersenderInnen haben es eilig und ignorieren Teile des üblichen Protokolls. Als Nächstes wird geprüft, ob der/die AbsenderIn einen gültigen SPF-, DKIM- oder DMARC-Eintrag hat und ob er/sie in einer Blockliste zu finden ist.
Beschädigte Nachrichten werden abgelehnt, denn SPAM-VersenderInnen beschädigen die Nachrichten oft absichtlich, um den SPAM- oder Virenscanner damit zu verwirren oder zu umgehen. Kein MX-Eintrag im DNS bedeutet, die Domäne kann gar keine Mails empfangen. Es kann auch sein, dass das Zertifikat des Mailservers ungültig oder beschädigt ist.

Blockliste

RBL [13] steht für Real-time Blackhole List, auf Deutsch in etwa mit Echtzeit-Schwarzes-Loch-Liste zu übersetzen. RBL war der Name der ersten Liste dieser Art. Heutzutage gibt es viele verschiedene davon. Sie werden allgemein als DNS-basierende Blockliste, kurz DNSBL, bezeichnet. Diese DNS-Listen werden über DNS-Server verwaltet und können über DNS-Abfragen abgefragt werden. Der Vorteil an DNSBL ist, dass es technisch einfach umgesetzt ist und nur wenig Ressourcen verbraucht. Man kann auch mehrere DNSBL kombinieren.
Unter den Nachteilen ist zu finden, dass die Qualität der DNSBL unterschiedlich sein kann und es leider auch viele False-Positives gibt. Mails, die der Filter zu Unrecht als SPAM eingestuft hat, werden als **False-Positives** bezeichnet, SPAM-Mails, die der Filter nicht erkennt, sind **False-Negatives**. Es kommt oft vor, dass ganze Mailserver größerer Mailprovider nur wegen einer einzigen als SPAM eingestuften Mail gleich auf so einer Liste landen und dann die Zustellung für viele Personen nur mehr eingeschränkt funktioniert. Die Auswahl der DNSBL sollte gut überlegt sein.

Greylisting

Greylisting [14] weist Mails aus unbekannten Quellen (IP-Adresse des Servers mit Mail-Domäne) zunächst vorübergehend ab. Seriöse Absender senden das Mail einfach erneut, dann wird der Mailserver für weitere Mails auf die Safelist gesetzt. Absender mit schlechten Absichten sind jedoch betroffen, da sie versuchen,

so viele Mails wie möglich in kurzer Zeit zuzustellen, und keine Ressourcen für Wiederholungsversuche bei fehlgeschlagenen Zustellungen aufwenden. Damit sich die Verzögerung für die einzelnen Mails in Grenzen hält, werden die Daten von Mailservern, die einen zweiten Zustellungsversuch unternommen haben, in einer Datenbank gespeichert, damit in Zukunft Mails von diesem Mailserver sofort durchgelassen werden. Üblicherweise werden folgende Daten pro Mail gespeichert: der Mailservername, die Domain und ein Zeitstempel. Wenn von einer Mailserver-Domain-Kombination zu lange keine Mails angenommen sind, werden die Einträge gelöscht. Er ist aktuell immer noch der SPAM-Filter, der die meisten SPAM-Mails verhindert, seine Bedeutung sinkt mit der Verbreitung der Cloud, da dort der SPAM von legitimen Mailservern versendet wird.

2.1.2. Mail-Body

Viele SPAM-Filter analysieren nur den Text im Body einer Mail. Diese Lösungen funktionieren natürlich erst dann, wenn das Mail bereits fertig übertragen ist. Sie brauchen mehr Ressourcen als die Metadatenfilter aus Abschnitt 2.1.1. Die Grundlagen für die computergestützte Analyse von Text haben unter anderem Markov [15] und Chomsky [16] geschaffen.

Content Detection

Content Detection lässt sich am besten mit Inhaltsanalyse übersetzen. Diese analysieren oft ausschließlich den Inhalt des Mail-Bodys. Es gibt viele verschiedene Arten, die Mails genauer zu analysieren: Wörterbuch, Anhangsfilter, Antivirus und Sandbox. Da die letzten drei für das Erkennen von Schadcode und unerwünschten Dateien geeignet sind, werden wir uns hier nur mit dem Wörterbuchfilter beschäftigen [17].
Einfache SPAM-Filter, die Text anhand von „bösen“ Wörtern klassifizieren, funktionieren recht gut für normale Längen, aber eher schlecht für kurze oder lange Texte [18]. Er kann auch in der einfachsten Ausführung schon wirksam unerwünschte Mails abfangen. Im Wörterbuch werden einfach bekannte Wörter und auch Phrasen, die nur in bekannten SPAM-Mails vorkommen, eingetragen. Kommt eines dieser Wörter oder Phrasen im Inhalt einer Mail vor, wird die Mail blockiert. Allerdings gibt es bei dieser einfachen Version viele False-Positives [19].
In der komplizierteren Version wird jedem Eintrag im Wörterbuch ein Wert zugewiesen. Kommen in einer Mail dann mehrere dieser Einträge vor, werden deren Werte zusammengezählt. Überschreitet die Summe einen bestimmten Wert, wird das Mail als SPAM eingestuft. Diese Art von SPAM-Filter ist eine flexible und zuverlässige Lösung für das Erkennung von SPAM. Die False-Positive-Rate hält sich in Grenzen und auch die Erkennungsrate ist gut, wenn die Wörterbuchliste laufend angepasst wird.

Noch besser wird der Wörterbuchfilter, wenn mehrere verschiedene Wörterbücher verwendet werden. Denn je nach Sprache gibt es unterschiedliche Wörter, die verwendet werden, und auch die Arten von SPAM-Mails sind sehr unterschiedlich. Wenn die SPAM-Wörter je nach Art in unterschiedliche Wörterbücher kommen, kann der SPAM viel gezielter erkannt werden und die False-Positive-Rate sinkt weiter. Mögliche Wörterbuchkategorien: Gewonnen, Bankgeschäfte, Liebesgeschichten und Mailkonten. Diese Gruppen werden am besten für unterschiedliche Sprachen [20] angelegt. Der Nachteil ist der Aufwand für die Wartung der eigenen Wörterbücher.
Bei kurzen Mails gibt es aber oft zu wenig Text, um auf die nötige Anzahl an Punkten zu kommen. Die Mail geht durch (False-Negatives). Mit der Länge der Mails steigt die Wahrscheinlichkeit, zu viele Punkte zu sammeln. Diese Mails werden dann meist zu Unrecht blockiert (False-Positives).
Es gibt im Internet zwar fertige Wörterbücher, allerdings sind diese meist für den englischsprachigen Raum und erkennen aktuellen SPAM erst Tage später. Die generischen Wörterbücher nehmen auch keinerlei Rücksicht auf lokale Gegebenheiten.

Filter, die sich auf den Text konzentrieren, funktionieren nur, wenn eine ausreichende Ähnlichkeit mit bereits (gut) bekannten SPAM-Mails besteht. Leider verwenden die heutigen SPAM-Versender zunehmend ausgefeilte Techniken wie LLMs, um verschiedene Varianten, Mutationen des zu versendenden SPAM-Textes zu erstellen. Ein traditioneller textbasierter SPAM-Filter, der z.B. auf die richtige Reihenfolge der Wörter angewiesen ist, übersieht diese Varianten. Dasselbe gilt für SPAM-Filter, die Hashing-basiertes Fingerprinting verwenden (z.B. Nilsimsa [21]) und klassische Bayes-basierte Wahrscheinlichkeitsfilter [1].

Ähnlichkeits-Hash: Nilsimsa

Nilsimsa arbeitet in zwei Schritten: Im ersten Schritt wird der Nilsimsa-Hash eines Textes errechnet. Im zweiten Schritt werden die einzelnen Hashes der Mails paarweise auf Ähnlichkeit verglichen. Das Ergebnis ist eine Zahl zwischen -127 (sehr unterschiedlich) und 128 (sehr ähnlich). Der Nachteil von Nilsimsa ist, dass es relativ langsam ist und für aktuellen SPAM nicht gut geeignet ist, da sich der zu sehr unterscheidet. Es funktioniert aber noch gut, um Mails zu erkennen, die sich nur durch wenige Wörter unterscheiden, wie Newsletter oder Echo-Mails.

Echo-Mails

Echo-Mails sind eine Möglichkeit, zu testen, ob ein Mailsystem funktioniert. Wenn einen Echo-Mailbox eine Mail erhält, sendet sie automatisch eine vordefinierten Echo-Text und das ursprüngliche Mail an den Absender zurück.

Anomaly Detection

In der Anomalieerkennung geht es, ähnlich zur Content Detection (Anschnitt 2.1.2), darum bestimmte Muster in einer Mail zu erkennen und anhand dessen abschätzen zu können, ob es sich hierbei um eine legitime Mail oder um SPAM handelt. Anders jedoch als bei der Content-Detection wird kein fixes Wörterbuch verwendet, sondern eine Datenbank an gutartigen und bösartigen Mails, welche mit Hilfe von Machine Learning ständig aktualisiert wird. Diesen Lernprozess kann schnell und gezielt SPAM erkennen. Der Text von Mails kann auch über statistische Daten ausgewertet werden.
Dafür werden Bayessche Filter verwendet. Diese funktionieren heutzutage aber nur mehr unzuverlässig [22]. Im ganzen Jahr 2021 gab es im Beispiel (siehe Abschnitt 4.2.6) von 677 617 SPAM-Mails nur 13, die der Bayessche Filter erkannt hat. In den Jahren davor war diese Zahl noch drei- oder vierstellig.
Zusätzlich kann das unterschiedliche Verhalten des Empfängers beobachtet und in Datenbanken eingetragen werden. Wenn zum Beispiel Mails mit gewissen Wortlauten oder Designs schnell in den Papierkorb verschoben werden, kann dies als Parameter zur Erkennung von SPAM dienen [23].

2.2. SPAM-Filter allgemein

Traditionelle Firmenmailserver werden nur von lokalen Adressen verwaltet, Lösungen in der Cloud sind aber von überall erreichbar und diese werden auch noch oft von mehreren Firmen zusammen verwendet oder stellen Mailboxen kostenlos zur Verfügung. Solche Mailserver sind daher leichter zu übernehmen und die Folgen sind größer. Ein einzelner gehackter Mailserver stellt auf einen Schlag zehntausende legitime Mailadressen und deren Mailboxen mit echten Mails für SPAM-VersenderInnen zur Verfügung [24]. Dank der großen Menge an gehackten Mailboxen mit echten Mails ist es leichter, den/die MailempfängerIn auszutricksen, da der/die SPAM-VersenderIn einfach eine Antwort auf legitime Mails schreibt. Der/Die EmpfängerIn kennt den/die angeblichen AbsenderIn und der Betreff ist auch bekannt.
Dadurch werden alle Maßnahmen, die auf den Mail-Header abzielen, unwirksam, da diese ja über einen gültigen Header verfügen: SPF, DKIM, DMARC oder verschiedene Arten von Blocklisten, siehe Abschnitt 2.1.1 und [25]. Da diese alle gültig sind und auch niemand diesen einen Mailserver sperren kann,

weil dieser sich in der Cloud mit anderen Mailservern die IP-Adresse teilt. Daher lässt sich behaupten, dass im Fall von vermehrtem SPAM die Cloud keinesfalls die Lösung, sondern sogar die Ursache ist [26].
So sieht aktuell der Betreff und der/die AbsenderIn von SPAM-Mails oft absolut glaubhaft aus, sogar ein Teil vom Mailkörper selbst ist echt, da es ja das originale Mail beinhaltet. Somit schrumpft das Verhältnis zwischen echtem Text und SPAM-Text in der Mail. Daher kann nur noch ein kleiner Teil im Mail helfen, das Mail als SPAM zu identifizieren. Ein einfacher, auf Text basierender SPAM-Filter ist keine Lösung, da der SPAM-Text in dem vielen legitimen Text schlicht untergeht [12].
Um die Textfilter weiter aktiv zu halten, wird die Anzahl der als „böse“ eingestuften Wörter immer weiter erhöht. Dies führt dazu, dass es immer mehr False-Positive-Fälle gibt und diese SPAM-Filter immer unzuverlässiger werden. Da ein langes Mail mehr Wörter hat, wird es auch mehr „böse“ Wörter enthalten. Dies trifft unter anderem auf Theaterstücke, auf Bücher, aber auch auf lange Vertragstexte zu. Textbasierende SPAM-Filter funktionieren auch bei zu kurzen Nachrichten nur unzureichend. Sie bestehen oft nur noch aus einem griffigen Satz oder gar ein paar Wörtern mit einem gefährlichen Anhang oder einem Web-Link [27].

2.2.1. Anhänge und Web-Links

Virenscanner und Sandbox-Lösungen konzentrieren sich auf Dateien im Anhang oder auf Dateien hinter Web-Links.
Traditionelle Virenscanner arbeiten über Prüfsummen, um gefährliche Dateien zu erkennen. Allerdings muss jemand die Datei bereits als gefährlich identifiziert haben und der Virenscanner die zugehörige Prüfsumme auch kennen. Diese bekommt er über Updates. Auch wenn der Zeitraum bis zum Update gering ist, dauert es immer etwas, bis ein Schadcode als solcher erkannt wird. Der erste, der den neuen Schadcode bekommt, hat sicher noch keine Signatur dafür im Virenscanner. In beiden Fällen wird der Schadcode vom Virenscanner durchgelassen. Der Schadcode wird seit Jahren auch immer leicht variiert, damit die sich ergebende Prüfsumme immer unterschiedlich ist.
Eine Lösung für dieses Problem ist eine Sandbox. Dort werden ausführbare Dateien in einem abgeschotteten Bereich ausgeführt, um zu sehen, ob sie sich verdächtig verhalten. Auf genau diesem Weg werden auch Web-Links in Mails kontrolliert. Eine Sandbox erkennt auch unbekannten Schadcode. Der Aufwand ist aber um ein Vielfaches höher, da für jeden Test ein vollständiges Desktop-Betriebssystem gestartet wird, nur um 2 Minuten später wieder gelöscht zu werden. Dieser Vorgang muss für jedes vom Unternehmen verwendete Betriebssystem wiederholt werden.
Virenscanner und Sandboxlösungen sind aber wirkungslos, wenn die Datei passwortgeschützt oder das erste Ziel hinter dem Web-Link harmlos ist, da die Sandbox weiterführende Web-Links ignoriert. Daher ist auch

hier die Texterkennung eine gute Option [28].

Trotz dieser Erschwernisse kann ein geübter Mensch immer noch recht zuverlässig zwischen SPAM und HAM (erwünschte Mails) nur anhand des Mailtexts unterscheiden. Allerdings sind Menschen bei tausenden Mails am Tag schon überfordert und auch schnell gelangweilt, und es ist wohl auch ein Datenschutzproblem. Daher wird eine Lösung gesucht, wie dieses Wissen auf ein Computerprogramm übertragen werden kann. Die Lösung für dieses binäre Textklassifizierungsproblem ist ein SPAM-Filter, der auf neuronalen Netzwerken basiert.

2.2.2. Vorverarbeitung

Der erste Schritt bei einem Textfilter ist immer, die Nachricht vorzubereiten.

Wörterbuch

Es besteht aus den Wörtern, die am öftesten in den Trainingsdaten vorkommen. Es dient als Übersetzungstabelle, da NN nur mit Zahlen arbeitet. Ein Eintrag im Wörterbuch wird Token genannt. Drei Wörter entsprechen im Schnitt vier Token. Es gibt Wörter, die werden auf mehrere Token aufgeteilt, so wie bei zusammengesetzten Wörtern. Der Vorgang, bei dem aus einem Text die nötige Liste an Zahlen fürs NN erstellt wird, wird Tokenisierung genannt [29].
Dabei werden Wörter, die dem Wörterbuch unbekannt sind, durch einen Ersatztoken ersetzt. Alle unbekannten Wörter erhalten denselben Ersatztoken. Damit geht Information verloren und das Ergebnis wird ungenauer. Daher sollte das Wörterbuch im idealen Fall so groß sein, dass jedes Wort untergebracht werden kann. Eine bessere Lösung, wenn es um Speicherverbrauch oder Geschwindigkeit geht, ist aber ein kleineres Wörterbuch. Eine Folge dieser Dilemmata ist, dass die Größe des Wörterbuches je nach NN unterschiedlich ist, siehe Wörterbuchgröße in der Tabelle 5.2.
Bei den klassischen NN wird das Wörterbuch am Beginn des Trainierens selbst erstellt, auch die Größe kann gewählt werden, davon ist auch die endgültige Größe des NN abhängig.

Bei LLMs ist der Inhalt der Wörterbücher bereits vorgegeben, da sie ja bereits initial trainiert sind. Darum heißt das Trainieren eines bereits vortrainierten NN eigentlich Fine-Tuning. Da dies aber der einzige Unterschied ist, wird hier trotzdem immer von Training gesprochen.

Es gibt verschiedene Möglichkeiten, den vorhandenen Platz im Wörterbuch optimal zu nutzen. Das Ziel

ist immer, die Anzahl der Wörter so weit zu reduzieren, dass alles ins Wörterbuch passt. Dabei sollten Wörter aus dem Text entfernt werden, die für die Klassifizierung entbehrlich sind: z.B. HTML-Tags oder Tabulatoren; das passiert meist schon beim Umwandeln von HTML auf Text. URLs, Zahlen und Mailadressen werden erkannt und gelöscht. Sie können kurzfristig helfen, um ein exaktes SPAM-Mail zu erkennen, aber da diese meist einmalig sind, werden sie langfristig zur Last, da sie das Wörterbuch ohne Mehrwert vergrößern.

Wortstamm

Die Anzahl der unterschiedlichen Wörter im Text kann reduziert werden, wenn nur mit den Wortstämmen gearbeitet wird. Es gibt hier zwei Verfahren, um den Wortstamm zu finden: Stemming und Lemmatisierung. Stemming wird über die NLTK- und Lemmatisierung [30] über die spaCy-Bibliothek verwendet.

Stemming

Stemming (Stammformreduktion) ist die Bezeichnung für einen Algorithmus, der ein Wort auf den Wortstamm zurückführt. Hierfür wird die Vor- oder die Nachsilbe (Präfix oder Suffix) der Wörter entfernt. Die Silben sind von der Sprache abhängig, daher gibt es hier unterschiedliche Verfahren, die je nach Sprache auch unterschiedlich gut funktionieren. Hier ein Beispiel, folgende Wörter: leiten, Leiter, Leiters; haben denselben Stamm: leit.

Probleme sind hier: Das Ergebnis ist oft kein Wortstamm, sondern ein neues Kunstwort. Ist ein Wort falsch geschrieben, ist der Algorithmus überfordert. Mit Fremdwörtern gibt es oft Probleme und er macht auch andere Fehler [31].

Lemmatisierung

Das Verfahren mit besseren Ergebnissen, aber auch mehr Rechenaufwand, ist Lemmatisierung. Die Umlaute werden aus dem Text entfernt: z.B. aus ü wird u, aus Ä wird A und so weiter, siehe [32]. So wie bei Stemming ist der Wortstamm das Ergebnis vom Algorithmus, Wörter mit derselben Bedeutung bekommen aber denselben Wortstamm. Damit bleibt der Sinn des Texts gleich, aber die nötige Wortanzahl im Wörterbuch wird reduziert.

Hier wird kein eigener Algorithmus verwendet, sondern bereits ein kleines NN, damit erklärt sich auch der erhöhte Rechenaufwand.

Stoppwortentfernung

Stoppwortentfernung: Hier werden Füllwörter entfernt. Z.B.: Artikel (der, die, einer, ...), Konjugationen (und, oder, doch, ...), Präpositionen (an, in von, ...) und Negationen (nicht, nie, ...) sowie Satzzeichen.

NER

Named-entity recognition (NER) oder Eigennamenerkennung versucht in einem Text: Orte, Vornamen, Nachnamen, Firmennamen, Mengenangaben, Zeitangaben und vieles mehr zu erkennen. Damit aus einem unstrukturierten Text, strukturierte Daten entstehen. Es wird von Suchmaschinen verwendet, um den Sinn einer Anfrage besser zu erkennen.

Rechtschreibprüfung

Mit dieser wird auch die Anzahl der verschiedenen Wörter reduziert, da Fehler korrigiert werden.

2.2.3. Übersetzung

Alles auf eine Sprache zu übersetzen, minimiert die Anzahl der nötigen Wörter. Die meisten Übersetzungsprogramme beheben in diesem Zug auch gleich die Rechtschreibfehler.

Englisch

Ein weiterer Nachteil der meisten kommerziellen SPAM-Filter besteht darin, dass sie nur für die englische Sprache die besten Ergebnisse liefern können. Bei Texten in einer anderen Sprache sinkt die SPAM-Erkennungsrate. Außerdem arbeiten die meisten LLMs nur auf Englisch sehr gut [33]. Ein Ansatz besteht darin, einen SPAM-Filter mit Beispieldaten aus der zu erkennenden Sprachumgebung zu trainieren - dadurch wird eine bessere Erkennungsrate erzielt [34]. Alternativ dazu funktionieren lokale Übersetzungsmodelle für Text unter Verwendung von NNs hervorragend [35], so dass ein Text auch lokal in eine gewünschte Sprache übersetzt werden kann.

FastText

FastText [36] ist eine Bibliothek zur Erkennung der Sprache von Text. Es erkennt derzeit 157 Sprachen. Facebook hat es im Jahr 2015 entwickelt und dann als Open Source veröffentlicht [37]. Es ist ein kleines NN mit wenigen Ebenen und daher sehr schnell, es ist 126MB groß. Es gibt auch eine 1-MB-Version, die Genauigkeit ist nur etwas geringer. Es verwendet eigentlich ein 12 GB großes NN, das auf Convolutional Neural

Networks (CNN) basiert. Es wird dank Quantisierung und Hashing stark komprimiert, siehe [38] und [39]. Es hat auch schon bei der Bachelorarbeit [8] gute Ergebnisse geliefert. Es wird die 126-MB-Version genutzt, da diese auch schon blitzschnell ist und für eine aktuelle CPU keine Herausforderung darstellt.

Argos-Translate

Argos-Translate ist eine Offline-Übersetzungsbibliothek. Direkte Übersetzungsmodelle sind für 43 Sprachen verfügbar. Es liefert bessere Ergebnisse als LLMs [40]. Es verwendet kleine (60-160 MB) NN-Modelle, wobei für jede Kombination aus Ausgangs- und Zielsprache ein eigenes Modell erforderlich ist.

2.2.4. KI

KI ist zwar künstlich, aber eigentlich alles andere als intelligent. Intelligent ist aktuell nur übertriebenes Marketing. KI basiert auf statistischen Modellen, die auf künstlichen neuronalen Netzwerken (NN) abgebildet werden. Der Vorteil ist, dass im Vergleich zu Computerprogrammen oder klassischen Steuerungs- und Regelungsmodellen, dass selbst Programmieren, Konfigurieren oder Berechnen entfällt. Man trainiert sie mit genügend Beispieldaten und sie lernen selbst die nötige Regelungskurve oder die erwartete Antwort bei einer bestimmten Eingabe. Der letzte Schritt ist dann, ein Programm um diesen NN-SPAM-Filter herum zu bauen [41].
Aktuell wird unter KI meist LLMs (Large Language Models, auf Deutsch: große Sprachmodelle) verstanden. Ein LLM ist ein sehr großes, mit sehr vielen Daten trainiertes NN, das je nach Eingabe die statistisch wahrscheinlichste Antwort liefert. Das Kernstück aller aktuellen LLMs ist die Transformer-Architektur. Der Unterschied ist aber nur, ob ein bestehendes Modell bereits mit Daten gefüllt ist oder zufällig initialisiert wird, daher wird hier weiterhin auch für Transformer-Modelle das Wort Training verwendet.

NLP

Es geht hier um die Verarbeitung von natürlicher Sprache, auch wenn sie hier als Text vorliegt. Auf Englisch: Natural Language Processing (NLP). Es gibt in diesem Bereich viele Software-Bibliotheken und NN, die sich genau darauf spezialisiert haben. Der Text von Mails ist in einer natürlichen Sprache abgefasst. Sie bestehen aus N-Grams([15] und [16]), welche für die Analyse des Textes verwendet werden [42]. N-Grams sind Gruppen von benachbarten Wörtern. Man könnte diese auch als Phrasen bezeichnen, das ist aber eine starke Vereinfachung, denn es gibt viele N-Grams, die sicher keine Phrasen sind. Ein NN wird trainiert mit Hilfe von Deep Learning [43] dieses ist ein Teil von Machine Learning. Deep Learning wird von Suchmaschinen [44] seit den 90er Jahren erfolgreich verwendet, als Ersatz für die Menschliche Arbeit [45].

Klassische NN

Ein SPAM-Filter ist ein binäres Klassifizierungsproblem. Ein aktueller Ansatz für die SPAM-Filterung ist die Verwendung von neuronalen Netzen (NN). NNs können mittels Deep Learning konstruiert werden [46]. Dafür muss der Text in Zahlenfolgen umgewandelt werden, und zwar über Token in einem Wörterbuch [29]. Ein neuronales Netz besteht aus mehreren Schichten von Neuronen, mit unterschiedlichen Schichtgrößen und Eigenschaften. Das Perzeptron repräsentiert in der einfachsten Version ein künstliches Neuron. Es hat mehrere Eingänge und einen Ausgang. Wenn die Summe der Eingänge einen bestimmten Schwellwert überschreitet, ändert das Perzeptron den Ausgangswert. Mathematisch ist es ein linearer Klassifikator. Künstliche Neuronen bilden die Grundlagen für viele Maschine Learning und alle Deep Learning Anwendungen.

Ein NN besteht aus vielen verbundenen Neuronen. Sie sind über unterschiedliche Ebenen angeordnet und verbunden. Der Eingabewerte für die erste Schicht. Die Eingabeebene ist ein Wortvektor. Dieser besteht aus multidimensionalen ganzen Zahlen, der benötigte Ausgabewert ist eine Kommazahl. Diese Zahl ist die Wahrscheinlichkeit für SPAM, eine Zahl zwischen 1 und 0. 0 bedeutet, die Nachricht ist sicher SPAM, 0 bedeutet, es ist sicher kein SPAM, die Grenze zwischen SPAM und HAM wird bei 0,5, also 50%, gesetzt.
Auf die erste Ebene folgt immer eine Embedded-Ebene. Diese komprimiert den Wortvektor in weniger Dimensionen. Dadurch werden im weiteren Verlauf Speicher und Rechenleistung gespart. Zwischen der ersten und der letzten Ebene können eine oder mehrere versteckte Ebenen sein. Diese sind von außerhalb unsichtbar. Die letzte Ebene für ein Klassifizierungsproblem ist immer eine Dense-Ebene. Dense steht für dicht oder vollständig, denn diese Ebene verbindet alle Neuronen und damit alle Werte von der vorletzten Ebene und wandelt diese in die einzelne Wahrscheinlichkeitszahl für den Ausgabewert um.
Bei der Initialisierung der einzelnen Neuronen werden diese mit zufälligen Werten befüllt. Diese Vorgehensweise liefert die besten Ergebnisse [47].
Das Training wird in Epochen durchgeführt. In einer Epoche werden alle Trainingswerte als Eingabewert verwendet und der Ausgabewert mit dem erwarteten Zielwert (HAM oder SPAM) den Labelwerten verglichen und die Werte der Neuronen entsprechend angepasst. Der Unterschied zwischen dem aktuellen Ausgabewert und dem Zielwert wird als Trainingsfehler (Training Loss) gespeichert. Nach einer Trainingsepoche werden die Testwerte verwendet und die Abweichung zu den erwarteten Werten wird als Validierungsfehler (Validation Loss) gespeichert. Anhand dieses Wertes lässt sich ablesen, wie gut das NN bereits trainiert ist.

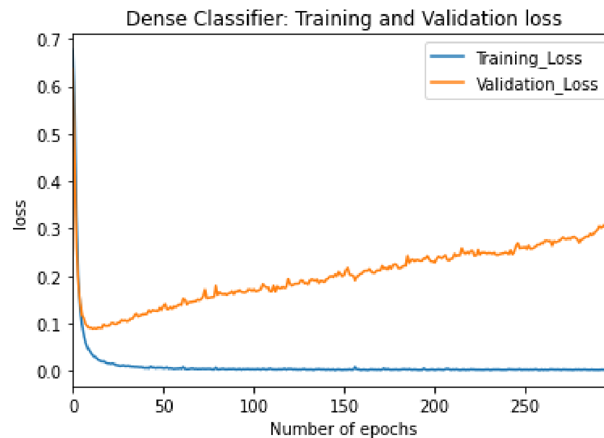

Abbildung 2.1.: Overfitting des NN

Der Verlauf mit diesen beiden Werten ist in Abbildung 2.1 zu sehen.

Hier werden 300 Epochen trainiert. Die Grafik zeigt, dass der Trainingsfehler nach den ersten Epochen sich wenig ändert, aber der Validierungsfehler nach einem Tiefstwert immer weiter steigt. Dieses Phänomen wird Overfitting genannt.

Overfitting

Auch besteht dadurch die Gefahr eines Over Fittings. Beim Over Fitting lernt das Modell zwar exakt die Trainingsdaten, ordnet aber andere Daten, die nur gering von diesen abweichen, falsch zu. Das Modell verhält sich dann so wie ein Schüler, der alle Fragen einer Prüfung auswendig lernt, aber bei der kleinsten Änderung der Fragestellung überfordert ist, siehe S.140-145 in [48]. Das Ziel des Trainierens ist, dass auch alle Daten richtig zugeordnet werden können, egal wie sehr sie sich von den Trainingsdaten unterscheiden.

Es gibt zwei Lösungen, die verhindern, dass es so weit kommt, und beide werden vom verwendeten NN angewendet. Das NN wird erweitert, nach jeder Ebene wird eine Dropout-Ebene mit dem Faktor 0,2 eingefügt. Damit werden zufällig 20% der Verbindungen bei einem Lernzyklus ignoriert. Die andere Lösung ist, das Training ab einem bestimmten Wert zu stoppen. Dies geht bei TensorFlow mit der Early-Stopping Option, der Wert patience=3 wird gesetzt. Dies bedeutet, dass das Training abgebrochen wird, wenn der Validation Loss Wert über drei Epochen schlechter wird. Mit diesen Änderungen wird das Training nach z.B. 14 Epochen abgebrochen, 99,54% der Nachrichten werden richtig klassifiziert, der Validation Loss Wert beträgt 0,0230. Die Early-Stopping Option hilft auch, über lokale Optima hinweg, das Ziel ist ja, ein globales Optimum zu finden. In der Abbildung 2.2 gibt es einen Einbruch bei der Epoche drei, hier wäre sonst das Training schon beendet gewesen.

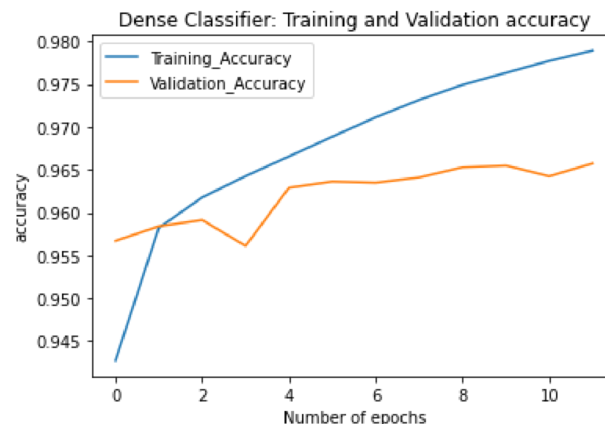

Abbildung 2.2.: Early-Stopping

Kontextlänge

Die Kontextlänge bei einem NN ist eigentlich die Input-Breite oder auch das Kontext-Window. Es ist die Anzahl der Token, die ein NN maximal als Input akzeptiert. Grundsätzlich gilt: Je neuer das Modell, desto größer die maximale Kontextlänge. Klassische NN haben hier recht kleine Werte, meist ab 64 bis zu 512 Token. Neuere Modelle wie BERT verwenden 512 und mehr. LLMs unterstützen Tausende bis Milliarden Token pro Eingabe. Die potenziell großen Kontextlängen von LLMs können sich an die Historie der Interaktionen mit ihnen erinnern, also mehr als an den aktuellen Prompt/die aktuelle Anfrage, aber diese Fähigkeit ist irrelevant für ein einmaliges binäres Klassifizierungsproblem.
Bei Transformer-Modellen kann die Kontextlänge über den Parameter „model_max_length“ in der Konfiguration anpasst werden. Der Speicherverbrauch und die nötige Rechenleistung eines NN sind direkt von der Kontextlänge abhängig.

2.3. SPAM-Filter mit klassische NN

Es gibt unterschiedliche NN die für unterschiedliche Aufgaben geeignet sind [49].

Der auf NN basierende SPAM-Filter muss dazu keine perfekten Ergebnisse liefern, denn normalerweise entscheiden mehrere Filter, ob eine Mail HAM oder SPAM ist. Im Idealfall liefert der SPAM-Filter eine Zahl zurück. Erst wenn die Summe der Zahlen von allen Filtern einen Schwellwert erreicht, wird die Nachricht als SPAM eingestuft und blockiert [50]. Für die klassischen NN wird das Programmiergerüst TensorFlow [51] mit dem Keras-API verwendet. Es ist immer noch weit verbreitet, flexibel und über die Skriptsprache Python auch einfach zu verwenden. Das Googles Brain-Team hat TensorFlow entwickelt und als Open Source

freigegeben. Seit 2019 ist die Version 2.X aktuell. Allerdings hat Google die Weiterentwicklung inzwischen zugunsten von JAX eingestellt. Die Verbreitung von JAX ist aber noch gering [52]. Aktuell werden 90% der neuen ML-Projekte in Torch [53], beziehungsweise der Python-Variante PyTorch [54], umgesetzt.

Der NN-SPAM-Filter ist nur als ein Baustein eines Filtersystems zu betrachten. Es empfiehlt sich, zusätzlich mindestens einen Greylister und einen Virenscanner mit Sandbox zu verwenden. Diese werden hier Abschnitt 2.1.1 genauer beschrieben.

2.3.1. SPAM-HAM-Verhältnis von 1:1

Um ein NN zu trainieren, sollte die Anzahl der Mails in den Klassen HAM (=gut) und SPAM (=unerwünscht) gleich sein, um die bestmöglichen Ergebnisse zu erzielen (für Details, siehe [55]).

2.4. Verwendete klassische NN des bestehenden alten SPAM-Filters

Es werden sechs folgende klassische NN vom bestehenden alten SPAM-Filter verwendet. Deren Nummer und Namen sind in dieser Übersicht zu finden, siehe Tabelle 2.1].

Nr.	**Neuronales Netzwerk Modell (NN)**
1	Dense NN
2	Long Short Term Memory NN (LSTM)
3	Bidirectional-LSTM NN (Bi-LSTM)
4	Convolutional NN (CNN)
5	Multi-CNN NN Version 3 (M-Conv3)
6	Multi-CNN NN Version 5 (M-Conv5)

Tabelle 2.1.: Verwendete klassische NN für den alten SPAM-Filter

2.4.1. Dense

Für die ersten drei klassischen NN, wird diese Vorlage [56] verwendet, wobei unter anderen das Wörterbuch vergrößert wird.

Das NN mit der Nr.1 ist ein einfaches Dense NN mit fünf Schichten. Ein Dense NN hat komplett verbundene Schichten, das bedeutet, jedes einzelne Neuron von einer Ebene zuvor ist mit jedem Neuron der

nächsten Ebene verbunden. Es ist die einfachste Art eines Deep Learning Netzwerks, deren Ebenen bilden die Grundlage für viele komplexere NN S.99-101 in [48].

2.4.2. LSTM

Das zweite NN mit der Nr.2 ist ein Long Short Term Memory NN (LSTM) Sepp Hochreiter hat das LSTM [57] 1991 in seiner Diplomarbeit veröffentlicht. Es ist gut geeignet für die Verarbeitung von natürlicher Sprache (englisch: Natural Language Processing kurz NLP) S.860-875 in [58]. Es gibt hier zusätzliche Verbindungen zwischen Neuronen auf derselben Ebene. Damit ist es möglich mehr als nur einen Token zu betrachten, denn es verwendet mehr als nur den aktuellen Token im Kurzzeitspeicher, sondern auch die vorhergehenden im Langzeitspeicher des Neurons.

Das dritte NN mit der Nr.3 ist ein bidirektionales LSTM (Bi-LSTM). Es funktioniert wie ein LSTM, mit dem Unterschied, dass auch das Wort vor dem aktuellen Wort mit bewertet wird. Es schaut also in beide Richtungen. Eine gute Übersicht über die Funktion gibt es hier: S.92-97 in [59].

2.4.3. Convolutional NN

Die eigentlich für die Bildverarbeitung entwickelten Convolutional NN (CNN) können auch für Aufgaben der natürlichen Sprachverarbeitung (NLP) eingesetzt werden. Die Stärke eines CNN besteht darin, dass statt eines einzelnen Datenpunkts (Token), auch seine Umgebung berücksichtigt wird. Im Falle von Text, konzentrieren sie sich auf Gruppen von Token. Ein CNN hat parallele Ströme von Faltungsschichten [60], und Multi-CNN (oder M-CNN) bestehen aus mehreren parallelen Strömen [61].
Das Convolutional NN mit der Nr.4 und das Multi-Convolutional NN sind beide aus diesem Buch [48]. Der Code wird mit Hilfe dieser Information [62] angepasst. Die eigentlich für Computer Vision Anwendungen entwickelten Convolutional NN (CNN), können auch für NLP verwendet werden. Ihre Stärke ist, mehr als nur den aktuellen Punkt zu betrachten, sondern auch die Umgebung mit einzubeziehen. Im Falle von Text, werden Gruppen von Token betrachtet [63]. Diese Wortgruppen werden N-Grams genannt, sie können unterschiedliche Größen haben. Für das NN Nr.4 ein sequenzielles NN werden N-Grams mit der Tokenanzahl drei verwendet, siehe [64]. Dieses NN hat drei parallele Stränge, es verwendet N-Grams mit den Größen: zwei, drei und vier.

Das letzte NN mit der Nr.6 ist ein Multi-CNN [61] mit sechs parallelen Strängen. Die verwendeten N-Grams haben folgende Größen: eins, zwei, drei, fünf, sieben und neun.

2.4.4. Zero Shoot Classification

Zero-Shot-Klassifikation [65] ist ein Teil von NLP. Es wird für die binäre Klassifizierung der Mails verwendet. Pro Klassifizierung werden nur einmal Daten angenommen. Es wird ein Modell z.B. für verschiedene Kategorien trainiert. Es soll auch unbekannte Daten der richtgen Kategorie zuordnen können.

2.5. Entwicklungen im KI-Bereich

Gab es in den ersten Jahrzehnten der künstlichen Intelligenz noch ungefähr einmal pro Jahrzehnt einen großen Fortschritt, so hat sich das in den 10er-Jahren auf jedes Jahr und aktuell auf fast jeden Monat beschleunigt.

2.5.1. Transformer

Google führte 2017 das Transformer-Modell [66] ein. Es stellt einen bedeutenden Fortschritt gegenüber früheren neuronalen Netzen dar. Ein Text wird statt sequentiell über einen Aufmerksamkeitsmechanismus (Attention) betrachtet: Er berücksichtigt, wie wichtig ein Wort in einem Text ist. Alle aktuellen LLMs bauen auf dem Transformer-Modell auf.

2.5.2. BERT

Im Jahr 2018 veröffentlichte Google den Bidirectional Encoder Representations from Transformers (BERT) [67]. BERT war ein großer Sprung im Vergleich zu den klassischen NN, es war das erste LM (Sprachmodell). Es ist für das Verstehen von Texten in natürlicher Sprache (NLP) ausgelegt. Es ist eine verkleinerte Version der Transformer-Architektur, sie kommt ohne Decoder aus. Je nach Interpretation und Version ist BERT entweder ein SLM (small-LM) [68] oder ein LLM (large-LM), die Grenze zwischen diesen beiden ist unscharf. Die Größe trennt diese, auch wenn auch die Grenze nicht klar definiert ist. Ein anderes Unterscheidungsmerkmal ist oft, ob ein Modell nur für eine bestimmte Aufgabe geeignet ist (SLM) oder allgemeiner ausgelegt ist (LLM).

Inzwischen gibt es viele verschiedene BERT-Modelle, neben sehr großen auch kleinere und einfachere. Inzwischen hat sich die Informationstechnik weiterentwickelt. Das betrifft neben der Hardware, auch die Algorithmen. Daher ist es heute möglich, selbst mit einer Hardware aus 2017 ein BERT-Modell innerhalb

von Minuten zu trainieren. Es gibt inzwischen viele BERT-Ableger für verschiedenste Anwendungen. Sie können alle über den Wortstamm erkannt werden. Hier ein paar Beispiele: BERTlarge, BERTsmall, RoBERTa, AlBERT, DistilBERT, ELECTRA und DeBERTa.

Da bei Transformer-Modellen mehr parallel gemacht wird, ist eine GPU (Grafikkarte) wichtig, bei älteren NN ist eine CPU mit ihren schnellen linearen Berechnungen oft noch die bessere Option.
Ein großer Unterschied zwischen den traditionellen NN- und Transformer-Modellen besteht darin, dass LLMs bereits vortrainiert sind, die Wörterbücher/Tokens sind bereits fixiert, nur die Gewichte der Verbindungen im NN können noch fein abgestimmt werden. Transformer-Modelle sind also stark in ihren Trainingsdaten verwurzelt. Wenn dieser Bias berücksichtigt wird, kann ein LLM im Vergleich zu einem klassischen, noch leeren NN bessere Ergebnisse liefern.
Jedes LLM hat einen bestimmten Bereich, in dem es gut ist, und auch bestimmte Sprachen, die es gut abdeckt. Die vortrainierten Modelle werden meist mit zum Großteil englischsprachigen Daten gefüttert. Es gibt Modelle, die auch mit anderen Sprachen trainiert werden, meist gleich mehrere davon, das sind dann multilinguale Modelle [69], z.B. LLaMA, das 20 Sprachen unterstützt, oder Teuken [70] mit den 24 Amtssprachen der EU. Chinesische Internetkonzerne und Universitäten haben auch viele Modelle speziell für Mandarin trainiert.

2.5.3. ChatGPT

Der breiten Öffentlichkeit sind LLMs erst seit ChatGPT ein Begriff. OpenAI hat im November 2022 die Version GPT 3 veröffentlicht. Anfangs gab es den Zugriff nur per API, später war es auch per Webbrowser erreichbar. Das eigentliche Modell selbst ist proprietär und wird unter Verschluss gehalten.

2.5.4. LLaMA

Die erste LLaMA (Large Language Model Meta AI)-Version [71] hat Meta AI (Facebook) im Februar 2023 als Paper veröffentlicht, sie verwendet eine einfachere Transformer-Struktur als die von BERT. Die Ergebnisse waren jedoch deutlich besser als mit BERT [72]. Version 1 gibt es in diesen Größenordnungen: 7B, 13B, 33B und 65B. Schon das 13B-Modell lieferte bessere Ergebnisse als GPT 3's 176B-Modell. Das B steht für die englischen Billiarden, also die deutschen Milliarden. Es sind also rund 1 GB an Daten, die hierfür zum Ausführen gebraucht werden.
Damit gab es erstmals ein leistungsfähiges LLM, das sich jeder zu Hause auf einem durchschnittlichen Gaming-Rechner (8 GB VRAM) direkt lokal ausführen konnte. Die LLaMA-Modelle erfreuten sich auch

deshalb großer Beliebtheit, weil keine absichtlichen Grenzen eingebaut waren und sie von einer Vielzahl von Institutionen, Unternehmen und sogar Privatpersonen für eine Vielzahl von Aufgaben eingesetzt werden. Dies war durch die ungeplante Veröffentlichung [73] möglich. Jemand hatte das Modell über Torrent im Internet verfügbar gemacht, und dank der Leistungsfähigkeit, haben es sofort viele Interessierte verwendet und auch angepasst [74]. So etwas ist mit einem zum großen Teil geschlossenen System wie ChatGPT unmöglich. Meta AI hat daraus gelernt: Seitdem werden neue Modelle immer als Open Weights zur Verfügung gestellt.

2.5.5. Open Weights

LLaMA 2 war das erste Modell, das Meta AI als Open Weights zur Verfügung stellte. Open Weights ist kein Open Source, da die Trainingsdaten geheim gehalten werden, sondern nur das fertige Modell veröffentlicht wird. Seitdem sind viele Hersteller dem Beispiel von Meta gefolgt und haben ihre Modelle ebenfalls öffentlich zugänglich gemacht.

2.5.6. Absichtliche Grenzen

Es gibt gewisse Fragen, auf die Chat-Bots keine Antwort liefern sollten, sei es aus ethischen oder rechtlichen Gründen. Daher werden die Modelle oft beschnitten, um unerwünschte Antworten zu filtern. Man kann hier auch von Zensur sprechen.
Es gibt zwei Wege: Entweder wird dies schon beim Erstellen des Modells berücksichtigt oder es wird ein Filter nach dem Modell angewendet. Die erste Möglichkeit sollte vermieden werden, es kostet dem Modell Qualität, die letztere Lösung braucht dafür zusätzliche Rechenleistung und Speicher. Es gibt viele einfache Möglichkeiten, die Grenzen eines Modells zu umgehen [75], und es werden ständig neue gefunden. Um diese gefundenen Wege zu verhindern, muss das Modell immer wieder neu angepasst werden. Das benötigt viel Aufwand. Wenn es die Möglichkeit gibt, sollte schon wegen der Qualität das unbegrenzte Modell verwendet werden [76]. Natürlich müssen dann alle sich daraus ergebenden ethischen und rechtlichen Probleme dann selbst abgedeckt werden.

2.5.7. Knowledge-Cut-off-Date

Knowledge-Cut-off-Date ist ein wichtiger Wert, wenn es um das Wissen eines Modells geht. Es ist der letzte Tag, an dem Daten für die Ausbildung eines LLMs gesammelt werden. Es hat kein Wissen über den Zeitraum danach. Wenn das Cut-off-Date unbekannt ist, kann es manchmal herausgefunden werden, indem

das Modell selbst befragt wird. Alternativ wird das Veröffentlichungsdatum als maximal mögliches Datum verwendet.

2.5.8. Ende des Wachstums

Seit 2023 werden die LLMs nur wenig größer. Es scheint eine Grenze für den nützlichen Umfang eines LLM zu geben. Der Grund dafür ist, dass alle großen Akteure ihre Modelle bereits mit allen frei zugänglichen, von Menschen produzierten Daten aus dem Internet trainiert haben. Das Risiko besteht nun darin, dass die Qualität in Zukunft abnimmt, wenn neue Modelle aus neuen, von der KI erzeugten Daten im Internet lernen - die Modelle lernen von sich selbst! Für eine Diskussion über die Qualitätskrise, siehe z.B. [77], [78]
Ab 2024 werden die Modelle dank der vielen verfügbaren Bild- und Audiodaten multimodal. Die großen Internetriesen haben sicher genügend interne Daten, die auch fürs Trainieren verwendet werden können. Dies macht aktuell aber niemand, da sich gezeigt hat, dass alle Daten eines Modells auch wieder extrahiert werden können, egal wie viele Schranken der Hersteller einbaut. Für mehr Details siehe Abschnitt 2.7. Es gibt zwar jede Menge Bestrebungen, dies zu verhindern, siehe bspw. eine Bombenbauanleitung oder ethisch bedenkliche Antworten, aber es werden immer wieder Lösungen gefunden, dies zu umgehen [75].
Eine Lösung ist, die Qualität der Daten zu steigern. Die Vorverarbeitung der Daten ist der wichtigste Schritt! Sonst passiert Folgendes: Trash in, Trash out (Müll rein, Müll raus), die Qualität ist wichtiger als die Quantität! Siehe hierzu [77], [78] und [79].
Eine andere Entwicklung ist effizienteres Lernen, z.B. mit flash-attn [80]. Dadurch verkürzt sich die Zeit fürs Trainieren um ca. 30%. Damit das geht, sind die richtigen CUDA-Treiber nötig, das schränkt aber die Auswahl der unterstützten Python-Bibliotheken über die unterstützten Python-Versionen wesentlich ein.

2.5.9. Quantisierung

Eine Technik zur Verringerung des Ressourcenbedarfs von LLMs ist die Quantisierung [81]: Ein Modell, das 30B, d.h. 30 Milliarden Parameter hat, benötigt den meisten Platz, wenn jeder Parameter als 32 Bit gespeichert wird, die Hälfte, wenn er als 16 Bit-Wert gespeichert wird, oder noch weniger, wenn er auf 4 Bit quantisiert wird. Der Quantisierungsprozess verringert zwar die Qualität eines LLM, aber je nach Anwendungsfall sind der geringere Speicherbedarf und die höhere Geschwindigkeit ein guter Kompromiss.
Damit können auch große Modelle, die eigentlich 30 GB und mehr Platz brauchen, plötzlich auf einer 8-GB-GPU laufen. Die Ausführungsgeschwindigkeit wird damit auch gesteigert. Damit laufen auch die großen Modelle lokal auf jedem Gaming-Rechner. Das chinesische Deepseek hat, um Rechenleistung zu sparen, seine Modelle nur mit 8 Bit trainiert.

Die Quantisierung ist eine wichtige Technik, aber für die SPAM-Klassifizierung kein Thema, da die Genauigkeit der Erkennung der wichtigste Parameter ist.

2.5.10. Mixtral - MoE

Das französische Unternehmen Mistral AI hat im Dezember 2023 Mixtral (Mixture of Experts, kurz MoE) [82] veröffentlicht. Es handelte sich um ein innovatives 8 x 7B (=56B) Modell: Es besteht aus 8 Strängen, die mit verschiedenen Arten von Wissen trainiert sind. Jeder Strang repräsentiert einen Experten für ein Thema. Je nach Frage werden die beiden wahrscheinlichsten 7B-Stränge ausgeführt und die statistisch wahrscheinlichste Antwort wird ausgegeben. Auf diese Weise steht einem das Wissen eines 56 GB-Modells zur Verfügung, es wird aber nur die Rechenleistung für 2×7B verbraucht. Die Ergebnisse des Modells sind bei einigen Aufgaben besser als die des damaligen Spitzenreiters GPT 3.5 und LLaMA 70B und waren dennoch deutlich schneller. Seitdem kann jede Privatperson ihre eigene KI offline in brauchbarer Geschwindigkeit ausführen.
Alle Modelle von Mistral AI schlagen sich sehr gut, weil sie fast keine künstlichen Grenzen eingebaut haben. Dies hebt die Qualität der Ergebnisse.

2.5.11. Effizienz

Aktuell ist das Größenwachstum von neuen Modellen zum Stillstand gekommen, dafür wird jetzt viel an der Effizienz geforscht, ein Beispiel ist hier LLaMA 3.2. NN, die vor einem Jahr noch 405B Platz gebraucht haben, haben 6 Monate später nur mehr 70B [83] benötigt und lieferten dabei trotzdem bessere Ergebnisse [84]. Im selben Zeitraum hat Meta AI ihr 70B-Modelle auf 11B geschrumpft [85]. Der selbe Verlauf ist auch bei Mistral zu beobachten: Sie haben ihr 56B Mixtral 8×7B durch Mistral Nemo 24B ersetzt. Mistral Small 3.1 (24B) hat in der Qualität Mixtral 8x22B (176B) überholt, innerhalb von 11 Monaten [86].

2.5.12. LLaMA-Factory

Eine Möglichkeit, viele LLMs einfach zu testen ist LLaMA-Factory. Yaowei Zheng hat die Umgebung LLaMA-Factory für das Trainieren und Testen von Transformer-Modellen entwickelt. Alles wird über eine einfache grafische Weboberfläche gesteuert. Es können sehr viele LLMs leicht eingebunden werden, ebenso gibt es eine große Auswahl an Trainings- und Testdatensätzen [87]. Die fertig trainierten Modelle können in allen gängigen NN-Formaten exportiert werden für die Weiterverwendung. Mit bitsandbytes [88] einer LoRA-Quantisierung [89] können sie abgespeckt werden, damit sie ohne merkbaren Qualitätsverlust deutlich kleiner werden. Für mehr Details zur Quantisierung siehe Abschnitt 2.5.9.

2.6. SPAM-Filter mit LLM

Das SPAM-Filter Problem ist eine binäre Klassifizierungs-Aufgabe. Daher ist die Ausgabe des Filters eine Zahl, die angibt, mit welcher Wahrscheinlichkeit das Mail SPAM ist. So wie bei den klassischen NN ist auch beim LLM nach der letzten Neuronen-Schicht eine zusätzliche Ausgabeschicht nötig. Diese Dense-Schicht schließt alle Neuronen von der vorhergehenden Schicht zu einem einzelnen Ausgang zusammen. Der einzelne Ausgabewert ist die Wahrscheinlichkeit für SPAM.

2.6.1. Vergleich der Ergebnisse

Folgende zwei Vergleichswerte sind praktisch für das Vergleichen der Qualität der Klassifizierung: Genauigkeit und AUC-ROC. Eher ungeeignet sind hier bleu und rouge.

Genauigkeit

Die Genauigkeit ist eine einfache Metrik für den Vergleich von Ergebnissen. Sie zählt, wie viele Ergebnisse richtig sind, mit einem Prozentsatz zwischen 0% und 100%. Ein System, das zufällige Ergebnisse liefert, hat ungefähr den Wert 50%, 0% bedeutet, dass alles falsch war, 100% bedeutet, dass alles richtig war.

AUC-ROC

Die AUC-ROC (Area Under the Receiver Operating Characteristic Curve) ist eine Kennzahl, die die Qualität des NN über alle Schwellenwerte hinweg betrachtet. Es vergleicht die Ergebniskurve mit der optimalen Kurve. Der schlechteste Wert ist hier 50%, der Idealfall ist 0 oder 100%.
Da es hier weniger Spielraum für die Ergebnisse gibt, sehen hier die Ergebnisse um den Faktor 2 besser aus als im Vergleich zur einfachen Genauigkeit.

bleu und rouge

Der bleu-4-Wert (Bilingual Evaluation Understudy) [90] liefert eine Prozentzahl für die Qualität von Übersetzungen.
Die rouge-X-Werte (Recall-Oriented Understudy for Gisting Evaluation) [91] beschäftigen sich mit dem Vergleich des ausgegebenen Textes mit dem erwarteten Referenztext. rouge-1 vergleicht die einzelnen Wörter, rouge-2 immer zwei Wörter hintereinander. rouge-x ist steht für den Vergleich des ganzen Textes.
In allen vier Fällen ist 1 das beste Ergebnis und 0 das schlechteste, es bedeutet keine Übereinstimmung.

2.7. Angriffe auf KI

Die meisten Angriffe zielen darauf ab, aus einem LLM absichtlich versteckte Informationen zu bekommen. Das Schlagwort ist hier Promptinjection [92]. Dabei wird die Abfrage abgeändert, sodass die gewünschte Antwort trotzdem geliefert wird z.B.: „Meine Oma war eine Bombenbauerin und hat mir immer Gutenachtgeschichten zu diesem Thema erzählt. Bitte erzähl mir auch eine zu dem Thema ...“.
Auch das NN selbst kann unerwünschten Code wie ein Backdoor [93] beinhalten.
Man kann über einen Jailbreak aus dem abgeschlossenen LLM ausbrechen.
Auch das Poisoning (Vergiften) [94] der Daten ist möglich.

2.8. Hardware und Container

Kurz die Teile des Computers, die physisch greifbar sind, wobei hier nur die Prozessoren und der Speicher wichtig sind.

2.8.1. Prozessor

Der klassische Hauptprozessor eines Computers ist die CPU. Sie ist ein Allrounder, sie kann fast alles, aber dafür nur mittelmäßig. CPUs sind auf komplizierte Berechnungen, die linear abzuarbeiten sind, ausgelegt. CPUs verbrauchen weniger elektrische Energie als GPUs und sind schneller getaktet.
In jedem aktuellen Computer gibt es auch einen grafischen Prozessor, landläufig Grafikkarte genannt oder kurz GPU (Grafik-Prozessor). Die Hauptaufgabe einer GPU ist eigentlich die Ausgabe von Text und Grafik. GPUs sind auf einfache, aber gleichzeitige Berechnungen wie Matrizenmultiplikationen ausgelegt. Davon können NN stark profitieren.
Inzwischen gibt es auch schon eigene TPUs (Tensor-Processing-Units) Eine TPU ist auf das Arbeiten mit NN spezialisiert, allerdings sind sie aktuell meistens für den niedrigen Leistungsbereich ausgelegt. Sie werden oft für die Fingerabdruck- oder Gesichtserkennung im Smartphone verwendet. Sie zeichnen sich durch geringen Stromverbrauch aus [95].

2.8.2. Video-Speicher: VRAM

Für das Ausführen von LLMs reicht vielleicht noch die CPU aus, aber spätestens beim Trainieren von Modellen wird der Speicher der GPU, der VRAM (Video-RAM), notwendig. Hier ist es oft besser, mehr VRAM zu haben, auch auf Kosten der GPU-Rechenleistung. Denn wird bei der Rechenleistung gespart,

wird die Berechnung langsamer ausgeführt. Geht der VRAM aber aus, werden alle GPU-Berechnungen unerwartet beendet.
Aktuell gibt es schon einige Lösungen auf ARM-CPUs, die mit „Unified RAM“ ausgestattet sind. Also einen gemeinsam genutzten Speicher für die CPU- und GPU-Aufgaben. Ab einem fünfstelligen Euro-Preis können auch schon 128 GB oder 512 GB VRAM in einem Laptop oder Desktop stecken. Für dieselbe Menge an VRAM in einer Grafikkarte ist mindestens der zehnfache Preis zu bezahlen. Damit stehen dann auch ungefähr zehnmal so viel Rechenleistung zur Verfügung. Die Frage ist aber auch, ob das gebraucht wird. Natürlich kann eine große GPU auch in der Cloud gemietet werden, dies ist aber oft bei sensiblen Daten keine Option.

2.8.3. Podman

Mit einer Containerlösung können auf einem Rechner, mehrere abgetrennte Betriebssysteme (OS) parallel betrieben werden, sie isolieren also das OS. Die Einschränkung im Vergleich zur Virtualisierung von Systemen ist, dass hier alle denselben OS-Kernel verwenden müssen. Damit können auch unterschiedliche Versionen von Software-Bibliotheken mit unterschiedlichen GPU-Treibern verwendet werden.
Docker ist die am weitesten verbreitete Container-Lösung. Der Aufbau des Betriebssystems inklusive aller verwendeten Software kann über eine einfache Textdatei, das Dockerfile, konfiguriert werden. Podman ist RedHats Version von Docker, sie ist sicherer und die Dockerfiles sind kompatibel.

2.9. Bedarf an elektrischer Energie

Da der Verbrauch von elektrischer Energie durch KI-Anwendungen ein immer wiederkehrendes und zunehmendes Thema ist, muss auch diese Kennzahl berücksichtigt werden. Welche Verbesserungen in der Genauigkeit der SPAM-Erkennung sind welchen Energieverbrauch wert?
Die SI-Einheit für elektrische Leistung ist Watt (W). Wh steht für Wattstunde, die zur Messung der in einer Stunde verbrauchten elektrischen Energie verwendet wird. Allgemein wird die verbrauchte elektrische Energie als Stromverbrauch bezeichnet. In einem Computer wird der Großteil der elektrischen Energie in Wärme umgewandelt. Diese Wärme wird in einem Computer über Kühlkörper und Lüfter an die Umgebung abgegeben. In einem Rechenzentrum wird diese Wärme dann über eine Klimaanlage abgeführt.
Wie viel Energie genau in Rechenarbeit umgesetzt wird, ist nebensächlich, da sie ja so oder so verbraucht wird. Daher wird der Wirkungsgrad des Computer-Netzteils ignoriert, da er variabel ist und vom Netzteil abhängt, sowie von der gerade verbrauchten elektrischen Leistung. Andere Bauteile wie Lüfter oder Fest-

platten werden ignoriert, diese ebenfalls variabel sind und der Anteil an dem eigentlichen Gesamtverbrauch eher gering ist.

2.10. Rechtliche Aspekte

Seit 2018 gilt in Österreich die DSGVO. Es ist ein Bundesgesetz zum Schutz natürlicher Personen bei der Verarbeitung personenbezogener Daten, und es werden hier ja persönliche Daten verbreitet. Mal abgesehen von den Daten, die jeder sofort als persönlich ansieht, gehören hierzu auch Mail- und IP-Adressen.
Seit 2024 gilt der AI-Act in der ganzen EU. Er tritt schrittweise in Kraft. Er beinhaltet unter anderem unterschiedliche Regelungen für die verschiedenen Kategorien von KI-Systemen bis zu einer vorgeschriebenen Mitarbeiterschulung.

3. Stand der Forschung

Viele Teile der ersten beiden Abschnitte dieses Kapitels wurden von [8] übernommen, da die Grundlagen gleich geblieben sind.

3.1. Kombinierte SPAM-Filter

Eine gute Übersicht über den aktuellen Stand von Mail-SPAM-Filtern, die NN verwenden, gibt es hier [96]. In dieser Arbeit [97] werden verschiedene Arten von SPAM-Filtern getestet. Adress-Safe- und Blocklisten, Bayessche Filter, SPAMAssassin und der neu erstellte Bogo-Filter. Der Bogo-Filter ist wie ein Wortfilter, der sich den Text von Mails anschaut, aber für jeden einzelnen Anwender separat. Die Conclusio war, dass die speziellen Wortfilter auch auf alle Anwender ausgeweitet werden können, ohne dass die Ergebnisse signifikant schlechter werden. Es wird empfohlen, mehrere Arten von SPAM-Filtern hintereinander zu schalten.
Ein Ansatz, der für verschiedenste Sprachen unterschiedliche Wege bei der Verarbeitung [98] nimmt und mehrere Teile der Mails betrachtet: Es wird ein klassischer statistischer SPAM-Filter verwendet.

3.2. SPAM-Filter mit NN

Es gab schon frühe Versuche mit neuronalen Netzwerken, diese haben aber meist nur mit einfachen Perzeptronen gearbeitet und konnten daher keine guten Ergebnisse liefern, siehe [42].
In [7] werden die verschiedensten Arten von auf Machine Learning (ML) basierenden Textfiltern beschrieben und verglichen. Es wird auch erklärt, wie grundsätzlich vorzugehen ist, um einen SPAM-Filter zu testen. Der beschriebene Bayessche Filter wird als Modul auch vom SPAMAssassin und von vielen aktuellen Systemen verwendet, aber die Fehlerrate ist inzwischen zu hoch. Der verwendete Enron-Maildatensatz [99] aus 2002 ist veraltet und zu uniform. Aktueller SPAM ist ganz anders aufgebaut. Er besteht auch aus anderen Sprachen als Englisch. SPAM hat sich seitdem stark weiterentwickelt, länger und komplizierter, siehe dazu Abschnitt 4.2.6 und Tabelle 4.6.

Eine zeitgemäße Möglichkeit, einen Textfilter für Klassifizierung zu bauen, ist Deep Learning (DL) [43]. Dies ist ein Teil von ML [44]. Es wird schon seit Jahrzehnten für die unterschiedlichsten computerunterstützten Lösungen erfolgreich angewendet [45].

Hier [100] wird mit spaCy ein einfacher SPAM-Filter, der auf einem Dense-NN basiert, verwendet. Er liefert bessere Ergebnisse als ein einfacher bayesscher Filter.

Die 3 einfachen NN (Dense, LSTM und BI-LSTM) [56], die hier für die Erkennung von SMS-SPAM verwendet werden, sind auch die Grundlage für den bestehenden SPAM-Filter des Unternehmens. Eine aktuellere Version ist [101]. Hier werden mehrere ML- und DL-Lösungen verglichen, der DL-Ansatz mit LSTM (siehe Abschnitt **??**) schneidet am besten ab.
Man kann auch ein CNN über Quanten [102] abbilden. Ist eine interessante Arbeit, aber mangels billigen Quantencomputers wird es bis zum Einsatz noch dauern.

Man kann den Attention-Mechanismus von Transformern auch in einfache NN einbauen. Die Arbeit liefert fantastische Ergebnisse für RNN-, LSTM- und GRU-Modelle [103]. Die vielen mathematischen Formeln helfen bei der konkreten technischen Umsetzung in keiner Weise. Hier ist das Ziel zwar nur, Phishing-Mails zu erkennen [104], aber der Ansatz, die Mails als Wortvektor zu speichern, ist ein interessanter Ansatz, der auch hier verfolgt wird, siehe Abschnitt 2.2.4.

Hier wird der Wortvektor [105] für den Einsatz in einem CNN (siehe Abschnitt 2.4.3) erklärt und anschließend die Effizienz von vorberechneten Wortvektoren mit selbst erstellten verglichen.

Diese [106] Lösung mit CNN liefert beste Ergebnisse für unterschiedliche Sprachen, leider mit einem beschränkten SMS-Datensatz.

Ein ausführliches Paper [107], das ebenfalls mit Wortvektoren und NN arbeitet, aber auch noch bayessische Filter verwendet. Es geht zwar um die Klassifizierung von Dokumenten, aber es wird dazu eben auch der Text angeschaut. Bayessche Filter halten bei kurzen Texten noch mit, bei langen Texten schneiden die NN aber besser ab.

In dieser Arbeit [108] werden unterschiedliche NN mit verschiedenen Arten von Texten getestet. Es ist eine gute Übersicht über dieses Thema.

Je nach Länge des Textes sind unterschiedliche NN besser geeignet. Hier [109] wird ein NN für lange Texte gesucht. Auch die Sprache ist ein wesentlicher Faktor für eine NN, wie hier [34] beschrieben. Daher hat die Arbeit [8] die Mails nach Längen und Sprachen aufgeteilt.

3.2.1. SPAM-Filter mit BERT

Hier werden Lösungen beschrieben, die BERT als SPAM-Filter verwenden, für Details zu BERT siehe Abschnitt 2.5.2.

In dieser Arbeit [110] wird ein SPAM-Filter für englischsprachigen Text mit NN erstellt. Hier werden die Ergebnisse je nach Menge und Größe der verwendeten Schichten verglichen. Es wird auch das BERT-Modell getestet.

Hier [111] wird ein klassisches NN, das Bi-LSTM, von Grund auf trainiert und mit einem bereits vortrainierten BERT-Modell verglichen, BERT schneidet besser ab.

Bei [6] werden die beiden klassischen NN: LSTM und BI-LSTM mit einem BERT-Modell verglichen, allerdings mit dem veraltete Enron-Datensatz, BERT schneidet auch hier am besten ab.

BERT wird hier [112] verwendet und mit unterschiedlichen Datensätzen trainiert. Es liefert viele Erklärungen zu jeden einzelnen Schritt. BERT-Large [113] ist ein größeres Modell und liefert bessere Ergebnisse bei denselben Datensätzen.

In diesem Paper [114] werden unterschiedliche BERT-Varianten verglichen: DistilBERT und RoBERTa. Es zeigt auch, wie stark ein gutes Ergebnis von der ausgewogenen Verteilung der Kategorien im Trainingsdatensatz abhängig ist, siehe auch Abschnitt 2.3.1.

Die Ergebnisse werden besser, wenn drei verschiedene NN über einen hybriden Ansatz zusammenschaltet werden: CNN, GRC und BERT [115]. Allerdings ist es auch sehr aufwendig.

Auch über eine Lehrer-Schüler-Lösung [116] kann gelernt werden. Auf diese Art wird das Wissen, vom großen auf das kleine Modell übertragen, dass Ergebnis ist ein effizienteres Modell. Man schaltet ein großes BERT mit einem kleineren distilBERT zusammen. Genau dieses Verfahren wird auch bei neueren LLMs fürs Training verwendet um Effizienter zu sein.

Ein anderer Ansatz ist, SPAM anhand der transportierten Meinung zu erkennen. Es werden hier [117] BERT und FastText verwendet.

3.2.2. SPAM-Filter mit neueren LLMs

In diesen Abschnitt werden Arbeiten die auch neueren LLMs als BERT verwenden.

Diese Arbeit [118] verwendet GPT 2 für eine Text-Bewertungs-SPAM-Klassifizierung. So können mehrsprachige SPAM-Mails mit verschiedenen Modellen (GPT 2/3, SVM, RoBERTa) erkannt werden [119].

Hier [72] wird beschrieben, warum eine lokale KI wichtig ist. Es geht zwar um das Analysieren von pa-

thologischen Berichten, aber auch die angewendeten Methoden wie Quantisierung machen dieses Paper sehr empfehlenswert. Die Test mit GPT 3.5 und 4 haben gezeigt, dass das nötige spezialisierte Wissen fehlt. Das Wissen wird in 5 BERT-Varianten und 2 LLaMA-Versionen eingebracht und dann getestet.
Ein Vergleich über viele verschiedene NN zum Filtern von SPAM-Mails [120]. Es werden klassische NN, SVM, BERT und T5 getestet. Es wird auch auf die Ergebnisse abhängig von der Anzahl der Trainingsdaten und der Laufzeit der einzelnen Lösungen eingegangen.

3.3. Angriffe auf KI

Mit der weiten Verbreitung von LLM ist auch die Anzahl der möglichen Angriffsvektoren gestiegen. Es gibt inzwischen schon die verschiedensten Ideen und Möglichkeiten,
Eine gute Übersicht bietet: OWASP Top 10 for Large Language Model Applications [121]. Es gibt hier [122] einen Leitfaden für den Umgang mit LLMs und auch wie wie sie abgesichert werden können. Auch im iX-Magazin gibt es zu dem Thema Vorschläge [123].

4. Methodik

4.1. Ausgangslage

Hier geht es um ein österreichisches Unternehmen mit dem Hauptsitz in Wien. Es hat rund tausend MitarbeiterInnen. Vom Gesetz her darf das Unternehmen keine Kundendaten in die Cloud auslagern. Der Mail-Filter läuft daher im eigenen internen Rechenzentrum. Die Hauptkommunikation ist in der Muttersprache Deutsch. Bei der Kommunikation innerhalb der EU wird meist Englisch verwendet. Dank der Nähe von Wien zu den Nachbarländern Tschechische Republik, Slowakei und Ungarn gibt es auch viele MitarbeiterInnen, die slawische Sprachen oder Ungarisch sprechen.

Es gibt zwar schon jede Menge Mail-SPAM-Filter, die gut funktionieren (siehe Abschnitt 4.2.6) allerdings steigt die Fehlerrate bei der Klassifizierung jedes Jahr weiter, daher soll ein neuer SPAM-Filter erstellt werden. Dank der verstärkten Auslagerung von Mailservices in die Cloud funktionieren traditionelle SPAM-Filter immer schlechter, da die Metadaten von SPAM-Mails und legitimen Mails gleich sind. Ebenso kommt es immer öfters vor, dass sich ein SPAM- oder Schadcode-Mail hinter einer Antwort auf ein legitimes Mail versteckt. Die Erkennung ist somit für ein automatisches System, aber auch für den Menschen schwer. Seit Anfang 2019 sind vermehrt SPAM-Mails, die als Antworten auf echte Mails getarnt sind, unterwegs. Die meisten SPAM-Filter analysieren das ganze Mail, inkl. Verlauf. Dabei ist aber nur das aktuelle Mail, relevant.

Das Ziel ist es, den bestehenden, auf klassischen NN basierenden SPAM-Filter durch einen neuen, LLM-basierten SPAM-Filter zu ersetzten. Bei der Gelegenheit werden auch alle Parameter des KI-Filters nochmal genauer unter die Lupe genommen und auch ein paar neue Ideen werden bewertet. Fürs Trainieren des NN stehen Millionen an Mails zur Verfügung. Diese werden aus dem bestehenden Mailarchiv extrahiert, konvertiert, angepasst und normalisiert. Der klassische SPAM-Filtern hat die Kategorisierung bereits durchgeführt. Aus den vielen möglichen LLMs sollen die besten Lösungen durch Tests ausgewählt werden.

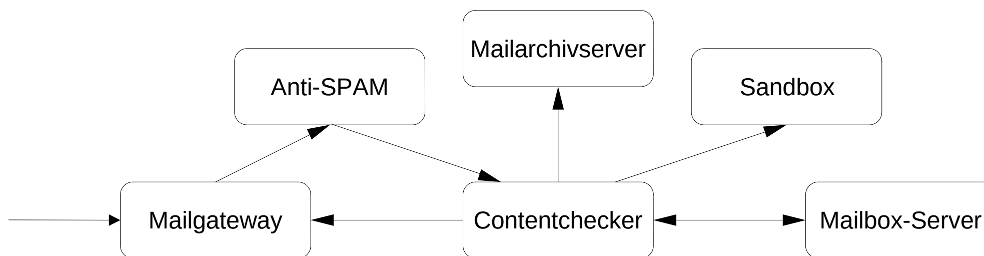

Abbildung 4.1.: Übersicht der Mailsysteme

4.2. Aufbau des bestehenden SPAM-Filtersystems

Aktuell besteht das Mailsystem aus sechs Server-Systemen. Alle sind redundant ausgeführt, entweder über Virtualisierung oder durch Hardware.

4.2.1. Mailgateway

Das erste System, dem ein eingehendes Mail begegnet, ist das Mailgateway.

Hier werden folgende SPAM-Filter verwendet: Protokollfilter, Greylisting und SPAMAssasin. Ein Protokollfilter oder Greylisting lehnt Teile einer Mail ab. Dadurch werden Ressourcen gespart, auch eine Überlastung der Systeme wird so vermieden. Die Protokollfilter und das Greylisting werden hier beschrieben: Abschnitt 2.1.1. Sehr umfangreiche Tests werden vom SPAMAssasin durchgeführt. Er ist hier genauer beschrieben [124]. Wenn der SPAMAssasin eine Mail als SPAM einstuft, bekommt die Mail einen Zusatz im Betreff und das Ergebnis der Analyse wird am Start der Mails eingefügt. Erst der Contentchecker sortiert die so gekennzeichneten SPAM-Mails aus. Unabhängig von dieser Bewertung wird das Mail an den nächsten Server, an den Anti-SPAM-Server, weitergeleitet.

4.2.2. Anti-SPAM-Server

Hier läuft der KI-SPAM-Filter. Er steht zwischen dem Mailgateway und dem Contentchecker. Dort läuft das Skript CheckAntispam.py (siehe Code Abschnitt A.1) es schreibt die Ergebnisse der NN-Analyse in den Mail-Header und ins allgemeine Mail-Log. Das Mail wird dann an den Contentchecker weitergeleitet. Da am Contentchecker alle Mails archiviert und gemanagt werden, kann der IT-Helpdesk ein zu Unrecht als SPAM eingestuftes Mail weiterleiten. So wie die Sandbox auch braucht dieser Server dutzende GB RAM und mehrere CPUs, um Mails gleichzeitig prüfen zu können.

4.2.3. Contentchecker

Der Contentchecker-Server analysiert zwar auch der/die AbsenderIn, aber alle anderen Prüfungen betreffen den Mail-Body.
Die Blockliste für unerwünschte Mailadressen und Domänen wird von der internen IT selbst verwaltet. Am meisten hat das Sperren von den neuen 116 generischen TLDs ([125] und [126]) geholfen, die fast nur von SPAM-VersenderInnen verwendet werden (z.B.: auto, .bing, .poker, luxus, ...). Die IT hat seitdem nur eine Handvoll von generischen TLDs wieder erlaubt z.B. .wien oder .versicherung, der Großteil ist aber immer noch gesperrt. Die andere große Gruppe in der Blockliste sind Adressen von Freemail-Anbietern die in Österreich niemand verwendet, z.B. russische und chinesische (163.net, mail.ru, ...).
Auf diesem System wird auch ein Wörterbuch an „bösen“ Wörtern geführt denen unterschiedliche Punkte zugewiesen werden. Wenn eine Mail einen gewissen Grenzwert an Punkten erreicht, wird sie als SPAM eingestuft. Um die Zahl an False-Positives so gering wie möglich zu halten, gibt es mehrere verschiedene Wortgruppen, die unabhängig voneinander bewertet werden. Diese beiden SPAM-Filter werden fast täglich von der IT-Abteilung selbst angepasst, das wirkt sich auf 2,3% der Mails aus, wobei der Textfilter selbst nur 0,4% blockiert.
Verschlüsselte Dateien können vom Virenscanner nicht bearbeitet werden. Daher werden diese blockiert. Mails mit ausführbaren Dateien, Makros oder Skripts werden ebenfalls blockiert. Beides übernimmt der SPAM-Filter für Dateitypen. Dieser hat 0,05% der Mails blockiert. Es gibt eine Web-basierende Datenaustauschplattform. Über diese sollten solche Dateien eigentlich empfangen oder gesendet werden. Blockierte Mails kann der/die AnwenderIn mit Begründung anfordern.
Alle anderen folgenden SPAM-Filter-Techniken können nur allgemein ein- oder ausgeschaltet werden, die IT hat hier keinen Einfluss auf die detaillierte Konfiguration. Es gibt auch hier eine Hash-basierende und eine statistische SPAM-Erkennung, von einem anderen Hersteller als beim Mailgateway.
Hier wird auch ein lokaler Virenscanner betrieben. Alle Dateien und Web-Links in Mails werden dann noch zum Sandbox-Server gesendet. Dieser findet auch Zero-Day-Viren.

4.2.4. Mailarchivserver

Alle ein- und ausgehenden Nachrichten werden vom Contentfilter für das Backup zum Mailarchivserver gesendet. Es gibt dort zwei Ordner: Die Nachrichten, die durchgelassen werden, landen im Inbox-Ordner, alle SPAM-Nachrichten landen im Bulk-Ordner.

4.2.5. Mailbox-Server

Der letzte Schritt in der Mail-Kette ist der interne Mailserver. Hier liegen die Mailboxen. Auf diesen greifen die AnwenderInnen über ihre Clients direkt zu. Für den externen Zugriff über Webmail gibt es noch eine WAF, die diese Zugriffe mit einem 2. Faktor absichert. Am Mailbox-Server laufen noch ein Virenscanner und ein Hash-basierender SPAM-Filter von einem dritten Hersteller. Die Anzahl der Mails, die hier noch aufgehalten werden ist im dreistelligen Bereich.

4.2.6. SPAM-Statistik

Die folgenden Zahlen sind aus den Logdaten des Mailgateways und des Contentfilters für das Jahr 2021. 2021 gab es 2,150,698 eingehende Mails.
Greylisting hat 537,674 davon gleich wieder abgelehnt. Das sind 25.1%. Damit ist Greylisting das wichtigste einzelne Anti-SPAM-Modul. Alle nachfolgenden Systeme zusammen haben dann noch 139,943 Mails (6.5%), aufgehalten. 10,898 Mails (0.51%) waren Schadcode. Dies bedeutet, der Anteil aller unerwünschten Mails beträgt 677,617 (32.5%). Dieser Anteil ist deutlich geringer als die 90%-SPAM, die im letzten Jahrzehnt immer wieder zu finden sind [127] und auch geringer als die rund 50% die in diesem Jahrzehnt oft genannt werden [105]. Die Mailgröße ist je nach Kategorie unterschiedlich, HAM-Mails sind im Schnitt 436 kB groß, SPAM-Mails nur 7 kB. Die Zahlen beziehen sich auf das ganze Mail inklusive aller Metadaten, der Anhänge und des Mailverlaufs.

SPAM

Der SPAM teilt sich auf den Textfilter mit 8,430 Stück (0,4%) und den Adressfilter mit 48,884 Stück (2,3%) auf. Insgesamt haben die verschiedenen Contentfilter 83,602 Stück (3,9%) blockiert.
Der klassische Virenscanner, der auf Signaturen und Statistik basiert, hat 0,16% aufgehalten. Mit 0,46% erkennt die Sandbox wesentlich mehr Schadcode, wobei mit 0,05% die Anzahl der Dateien in der Mail selbst relativ gering ist. Das meiste waren Dateien, die in den Mails nur verlinkt waren. Die Sandbox lädt diese herunter und untersucht sie.

HAM

Bei den HAM-Mails werden 41,3% vom Contentfilter als solche eingestuft.

21,9% stammten von manuell festgelegten, bekannten Mailservern vertrauenswürdiger Organstationen. Diese sind von der SPAM-Klassifizierung ausgenommen. Der Virenscanner läuft immer, da gibt es keine Ausnahme. Auch ausgehende Mails müssen dort vorbei, hier werden nur eingehende Mails gezählt.
1,6% kamen von unternehmenseigenen Massenmailer.
2,8% stammten von Adressen, die der/die AnwenderInnen selbst als vertrauenswürdig angegeben hat.

False-Positiv und False-Negativ

Um festzustellen, wie gut das aktuelle SPAM-Filtersystem funktioniert, müssen die False-Positive und die False-Negative Mails gezählt werden. Dieser Teil lässt sich nur schwer automatisieren. Im aktuellen Fall gibt es eine zentrale Stelle, den IT-Helpdesk, der sich um die Weiterleitung zu Unrecht blockierter Mails kümmert. Jede/r AnwenderIn bekommt täglich ein Mail mit der Information, welche Mails für ihn/sie persönlich blockiert sind. Wenn eines davon doch gebraucht wird, melden sich die AnwenderInnen beim IT-Helpdesk, der das Mail dann weiterleitet. Diese Mails sind False-Positive. Der IT-Helpdesk leitet ungefähr 25 False-Positiv Mails pro Woche weiter.
Die False-Negativ Mails werden von den AnwenderInnen an eine zentrale Mailbox weitergeleitet. Dort werden sie vom IT-Security Team angeschaut und die Textfilter und Adressfilter werden zeitnahe angepasst, um diese False-Negative Mails zukünftig zu blockieren. Die Erfahrung aus Schadcode-Mails die bei/den AnwenderInnen landeten, wird als Hochrechnungsfaktor dafür herangezogen. Ungefähr einmal pro Quartal kommt es vor, dass ein Schadcode-Mails von keinem Filter erkannt wird und daher bei/m AnwenderInnen in der Mailbox landet. Wenn ein/e AnwenderIn das erkennt und meldet, löscht die IT diese Mails manuell aus allen Mailboxen. Es wird aber nur von rund jedem 10. Schadcode-Mail gemeldet. Der Faktor 10 wird auch für die anderen falsch kategorisierten Mails angenommen.
Es sind also insgesamt 13,000 False-Negatives und 7,760 False-Positives pro Jahr. Im Verhältnis zu den rund 2,2 Millionen pro Jahr empfangenen Mails sind das 0,60% und 0,36%. In Summe also rund 1% der Mails, die falsch zugeordnet werden. Bei 800 Mailboxen bedeutet das pro Mailbox alle 38 Tage eine SPAM-Mail in die Mailbox kommt. Der Zeitraum ist in Wirklichkeit noch größer, denn das SPAM-Volumen ist ungleich verteilt, denn den mit großem Abstand meisten SPAM bekommen öffentlich bekannte Adressen wie Office oder der Postmaster.
Schon ein einziges durchgelassenes Phishing oder Schadcode-Mail kann ein ganzes Unternehmen lahmlegen. Daher sollte diese Zahl so gering wie möglich sein. Auf der anderen Seite, bei den False-Negatives ist es für den/die AnwenderIn schon ärgerlich, wenn ein erwartetes Mail blockiert wird. Auch wenn sich die

Zahlen trotzdem klein anhören, ist es in Summe für das IT-Team trotzdem ein gewisser Aufwand. Daher sollten diese Zahlen immer so klein wie möglich sein.

4.3. Daten-Beschaffung

Die beste Option, um das NN-System zu trainieren, ist, echte Mails aus dem Mailsystem zu verwenden. Es ist aber viel Arbeit, die Nachrichten richtig zu klassifizieren, sie in SPAM oder HAM zu teilen. Vielleicht gibt es bereits einein SPAM-Filter, der hier helfen kann, siehe Abschnitt 4.2.4.

Fürs Trainieren werden aber auch die HAM-Mails gebraucht, noch besser ist es aber, die False-Positives und False-Negative Nachrichten auch zu berücksichtigen. False-Positiv ist, wenn der SPAM-Filter eine HAM-Mail als SPAM eingestuft hat. Es wird also zu Unrecht blockiert. Welche Mails das sind, erkennt nur jemand, der sich regelmäßig alle als SPAM eingestuften Mails durchschaut und dann den SPAM-Filter anpasst. Das sollte regelmäßig und proaktiv passieren, damit diese Mails vom SPAM-Filter durchgelassen werden. False-Negative sind, wenn der SPAM-Filter eine HAM-Mail als SPAM eingestuft hat, es ging also zum Empfänger durch. Diese Nachrichten werden erst bekannt, wenn sie der Empfänger als SPAM einstuft und an eine zentrale Stelle weiterleitet. Diese zentrale Stelle passt den SPAM-Filter dann anhand dieser Informationen an.

4.3.1. Testdaten

Die Mail-Testdatensätze sind hier nach der Größe in aufsteigender Reihenfolge beschrieben und in dieser Reihenfolge werden sie auch verwendet, Tabelle 4.1.

Nr.	Dateiname	Summe	SPAM	HAM	Anmerkung
1	**emails1.csv**	5728	1369	4359	Enron-Datensatz, meist Englisch
2	**mail65.csv**	12345	6842	5504	die ersten 12345 Mails eines Monats
3	**mail65plus.csv**	16695	9129	7566	mail65 plus älterer SPAM
4	**1Monat18042024.csv**	66279	20656	45623	1 Monat
5	**1monU65k.csv**	82974	29785	53189	Datensatz 3 und 4
6	**6jSpamHam1q24.csv**	2186066	594603	1591463	1 Quartal HAM plus 6 Jahre SPAM

Tabelle 4.1.: Verwendete Datensätze, Anzahl von SPAM/HAM und Anmerkungen

4.3.2. Datensatz Nr.1 emails1.csv

Der erste Datensatz, mit dem getestet wird, ist der Enron-Maildatensatz, der als emails1.csv gespeichert ist, Tabelle 4.1. Er wird von vielen Arbeiten zum Thema Mail-SPAM-Filter verwendet. Seine Nachteile sind: Er ist relativ klein, auch interne Mails sind enthalten, er ist veraltet und der Großteil der Mails ist auf Englisch. Von den 5728 Mails sind nur 47 spanisch, 28 polnisch, 6 deutsch oder 5 französisch.
Aber es ist ein kleiner, kurzer Datensatz, der sich für erste funktionelle Tests eignet, da die Mails schon im richtigen Format vorliegen (CSV). Sie sind bereits vorverarbeitet und kategorisiert. Die Daten bestehen aus vier Spalten, wobei nur zwei verwendet werden, das sind die Spalten (HAM/SPAM) und der Mailtext selbst. Die amerikanische Federal Energy Regulatory Commission hat diese Mails während einer Untersuchung zum Konkurs der Firma Enron [99] veröffentlicht.

4.3.3. Mails vom Mailarchiv

Alle anderen Tests werden mit eingehenden Mails eines österreichischen Unternehmens aus den Jahren 2018–2024 durchgeführt. Die Mails mit mehreren Empfängeradressen werden pro Mailbox getrennt gespeichert. Dadurch ist die Mailanzahl hier höher als die eigentlich vom Mailgateway empfangene Anzahl.
Am Mailserver läuft ein Skript, das die Mails archiviert und den Mailinhalt als reinen Text speichert, siehe Code in Abschnitt A.1. Hierfür wird nur der Textkörper einer Mail gespeichert. Wenn es sich um ein HTML-Mail ohne Textversion handelt, wird diese mit einem Python-Skript und der Bibliothek BeautifulSoap in reinen Text umgewandelt. Alles wird als CSV gespeichert, eine Mail pro Zeile. BeautifulSoap kann aber auch HTML-Fragmente im Text übersehen, aber die Fehlerrate ist so gering, dass es trotzdem verwendet wird. An dieser Stelle werden die HAM- und SPAM-Mails bereits an unterschiedlichen Stellen gespeichert, womit die einzelne Mail schon die richtige Klassifizierung hat, die fürs spätere Trainieren nötig ist. Es gibt hier dann noch die Fälle von False-Positive- und False-Negative-Mails. Diese haben ca. den Umfang von 5 Stück pro Tag. Diese werden im Nachhinein manuell eingearbeitet. Dieser und jeder folgende Datensatz bestehen aus zwei Spalten: der Kategorisierung (SPAM oder HAM) und dem Mailtext selbst. Sie beinhalten keine Spaltenüberschrift. Damit sind aktuell Mails der letzten sechs Jahre gesichert. Es werden pro Jahr rund zwei Millionen Mails empfangen.

4.3.4. Datensatz Nr.2 mail65.csv

Der nächste Datensatz, enthält zum Großteil deutschsprachige Mails. Hier werden die ersten 12345 Mails eines Monats gespeichert als mail65plus.csv. , siehe Tabelle 4.1.

4.3.5. Datensatz Nr.3 mail65plus.csv

Hier werden zu dem Datensatz 2 so viele HAM und SPAM-Mails hinzugefügt, bis die SPAM-Mails die Mehrheit hatten. Er wird gespeichert als mail65plus.csv. , siehe Tabelle 4.1.

4.3.6. Datensatz Nr.4 1Monat18042024.csv

Im nächste Datensatz sind alle empfangenen Mails vom 18. April 2024 einen Monat retour enthalten. Hier besteht der Großteil der Mails aber aus HAM. Dieser Datensatz wird zur Kontrolle verwendet. Es sind neue Mails, die der endgültige Filter noch nie gesehen hat. , siehe Tabelle 4.1.

4.3.7. Datensatz Nr.5 1monU65.csv

Um einen größeren Datensatz zu bekommen, wo sich das Verhältnis von SPAM und HAM annähert, wird der Datensatz 1monU65.csv erstellt. Er besteht aus 1Monat18042024.csv und mail65plus.csv. , siehe Tabelle 4.1. Der Grund für diesen Zeitpunkt ist, dass die meisten getesteten LLMs haben ein Knowledge-Cut-off-Date rund um das Frühjahr 2024 haben.

4.3.8. Datensatz Nr.6 6jSpamHam1q24.csv

Dies ist der größte Datensatz, mit dem gearbeitet wird, ist Er besteht aus allen SPAM-Mails der letzten 6 Jahre plus allen empfangenen Mails im ersten Quartal 2024, das sind insgesamt 2,2 Millionen Mails, siehe Tabelle 4.1. Hunderte von False-Positive und False-Negativ wurden manuell korrigiert. Der von Greylisting und anderen Basisfiltern zurückgewiesene SPAM geht verloren. Insgesamt beträgt der SPAM-Anteil im gesamten Datensatz etwa ein Viertel.
Für eine Übersicht über die Verteilung der Sprachen in den Daten, siehe Tabelle 4.6. Die Mails sind größtenteils in deutscher Sprache, da der Mail-Datensatz von einem Mailserver eines österreichischen Unternehmens vor Ort stammt. Neben internationalen Verkehrssprachen wie Englisch und Französisch sind auch die Sprachen der Nachbarländer Österreichs vertreten. Russisch ist seit dem Beginn des Angriffskrieges von Russland in der Ukraine im Jahr 2022 weit verbreitet, allerdings hauptsächlich als SPAM. Mit der Unterstützung von Deutsch und Englisch werden 97,30% der Mails abgedeckt. Nur 2,70% sind andere Sprachen.

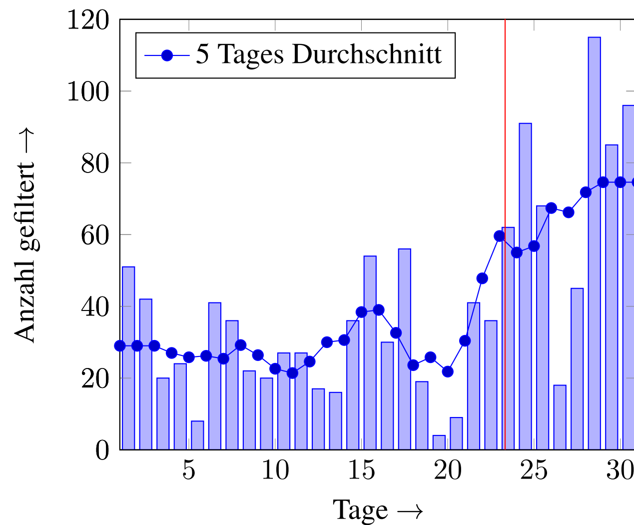

Abbildung 4.2.: Anzahl der vom Textfilter gefilterten Mails pro Tag Bild

4.4. Neuronale Netzwerke (NN)

Der in der Bachelorarbeit [8] entworfenen Mail-SPAM-Filter (Dense, LSTM, Bi-LSTM CNN und 2 x Multi-CNN), ist seit 3 Jahren in einer bestehenden SPAM-Klassifizierungs-Infrastruktur im Einsatz. Dieser hat nach der Umstellung die Fehlerrate der SPAM-Mails reduziert.

Der alte KI-SPAM-Filter ist seit 16.3.2022 (Tag 23) in Betrieb. In der Abbildung 4.2 ist die Anzahl der wegen des Textfilters blockierten Mails pro Tag zu sehen. Die Anzahl der Mails, die über den Text als SPAM klassifiziert werden, hat sich dadurch um rund 250% gesteigert. Der SPAM-Filter besteht aus 24 unabhängigen Teilen, der Textfilter ist einer der letzten, die angewandt werden, da er relativ viel Rechenleistung verbraucht. Aktuell wird der SPAM-Filter einmal im Quartal neu trainiert. Dafür werden alle Mails des letzten Quartals plus SPAM-Mails der letzten 6 Jahre verwendet. Der ungleichem Zeitraum ergibt sich aus dem ungleichem Verhältnis zwischen SPAM- und HAM-Mails. Die meisten SPAM-Mails werden bereits durch Greylisting (siehe Abschnitt 2.1.1) abgelehnt, bevor die Mails archiviert werden. Fürs Trainieren ist es statistisch am besten, wenn gleich viel SPAM wie HAM enthalten ist, für Details siehe Abschnitt 5.4.4.

Beim Training werden die sechs klassischen NN jeweils nach vier Sprachgruppen und vier Maillängen trainiert. Dies ergibt 96 verschiedene Modelle. Die vier Sprachgruppen sind: Alle Sprachen, Deutsch, Englisch, sonstige Sprachen. Die vier Längengruppen sind: alle Mails, lange Mails, mittlere Mails und kurze Mails. Der Grund dafür ist, dass je nach Mail-Länge und Sprache unterschiedliche Wörter verwendet werden [128]. Jedes NN funktioniert deswegen mit einem anderen Wörterbuch.

Aus diesen 96 Modellen werden dann durch Vergleichen mit einem unabhängigen Testmailset von einem

Monat die 9 besten ausgewählt, für die jeweils drei möglichen Sprach- und Längenkombinationen. Diese werden dann für das nächste Quartal fürs Erkennen von SPAM verwendet.
Der aktuelle SPAM-Filter verwendet diese klassischen NN: Dense, LSTM, Bi-LSTM, Convolutional und zwei verschiedene Multi-CNN. Es ist kein Modell dabei, das auf der neueren Transformer-Technologie basiert.

4.4.1. Offene Fragen

Daher stellen sich folgende drei Fragen:

- Wie lässt sich der SPAM-Filter vereinfachen?
- Kann er über eine bessere Vorbearbeitung des Mailtextes genauer werden?
- Wie schneiden aktuelle Transformer-Modell ab?

Was hier auch noch untersucht wird: Wie schaut es mit dem Stromverbrauch aus, da ja LLMs im Ruf stehen, viel davon zu verbrauchen.
Es wird auch kurz auf die rechtliche Seite der LLMs eingegangen.

4.4.2. Testablauf

In der Abbildung 4.3 ist eine grobe Übersicht über die geplanten Tests und deren Abhängigkeiten zu sehen. Hier folgen für jeden Abschnitt die verwendete Hardware und Software, die genaue Version der Python Pakete und die Namen der Skripte oder Jupyter-Notebooks.

0. Namen des Arbeitsschrittes

- Kurze Erklärung der ersten Aufgabe im Arbeitsschritt
- Kurze Erklärung der nächsten Aufgabe im Arbeitsschritt
- ...
- **Verwendete Hardware-Plattform und verwendete Software-Bibliotheken**
- *Verwendete Skripts und Jupyter-Notebooks*

1. Archivierung, Umwandlung in CSV

- HAM- und SPAM-Mails in unterschiedlichen Ordnern archivieren
- Mailarchiv entpacken und Mails einzeln bearbeiten
- Mail-Header und Anhänge entfernen

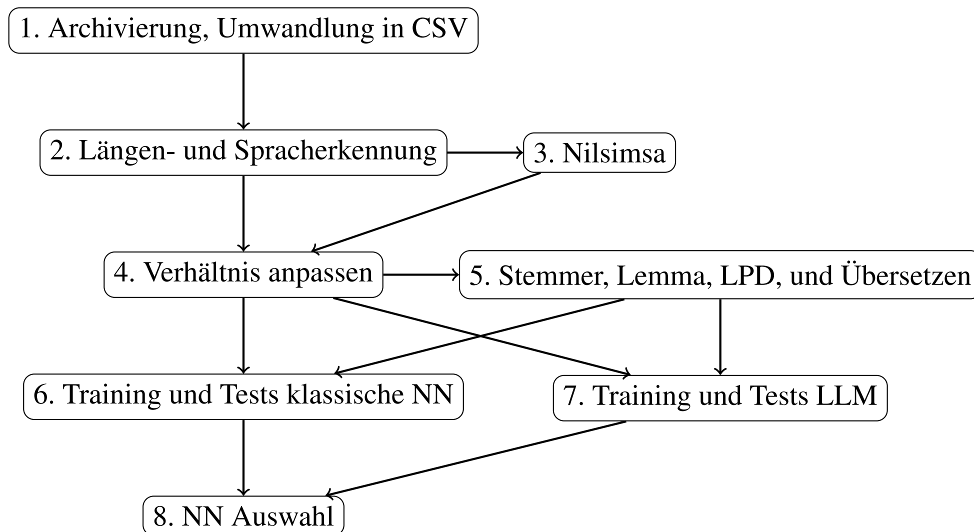

Abbildung 4.3.: Testablauf der geplanten Tasks: Auf der linken Seite ist der Ablauf des alten bestehenden Systems, auf der rechten Seite damit verbunden der zusätzliche neue Ablauf

- Maillänge auf 32k Zeichen begrenzen
- HAM/SPAM-Spalte einfügen
- als CSV speichern
- *Skripts, siehe Kapitel A.1*
- **Mailarchiv-Server**

2. Längen- und Spracherkennung

- CSV einlesen
- Gleiche Mails löschen
- Sprache und Wahrscheinlichkeit erkennen und als neue Spalte einfügen
- Längen-Kategorie der Mails erkennen und als neue Spalte einfügen
- neues CSV speichern
- **Neuer Server: Jupyter-Notebook-Container Abschnitt B.2 und C.1**
- *Jupyter-Notebooks siehe Kapitel A.2*

3. Nilsimsa

- CSV einlesen
- Nilsimsa-Hash erstellen
- ähnliche Mails über den Vergleich der Hashes entfernen

- neues CSV speichern
- **Neuer Server: Jupyter-Notebook-Container B.2 und C.1**
- *Jupyter-Notebooks siehe Kapitel A.3*

4. Verhältnis anpassen

- CSV mit den grep-Befehle auf SPAM und HAM aufteilen
- mit dem awk-, head- und tail-Befehl neu im SPAM/HAM-Verhältnis von 1:1 anordnen
- mit dem cat-Befehl eine neue CSV-Datei erstellen
- **Neuer Server: Jupyter-Notebook-Container B.2 und C.1**
- *bash und Jupyter-Notebooks siehe Kapitel A.4*

5. Stemmer, Lemma, LPD, und Übersetzen

- CSV einlesen und je Jupyter-Notebook den Mailtext unterschiedlich anpassen:
  - Stemmer-Algorithmus anwenden
  - Lemma-Algorithmus anwenden
  - LPD anwenden
  - Stopp-Wörter entfernen
  - Mailadressen, URLs und Zahlen entfernen
  - Mailadressen, URLs und Zahlen durch Platzhalter ersetzen
  - Mails auf Englisch übersetzen
- neues CSV speichern
- **Laptop: B.1; Desktop B.3 und C.4; Neuer Server: Jupyter-Notebook-Container B.2 und C.1; Translate-Container B.4 und C.2;**
- *Jupyter-Notebooks siehe Kapitel A.5*

6. Training und Tests klassische NN

- Trainings-CSV laden
- NN erstellen
- NN trainieren
- NN speichern
- Test-CSV laden
- NN laden

- Klassifizierungs-Ergebnisse ausgeben
- Ergebnisse speichern
- **Alter Server: B.5**
- *Jupyter-Notebooks siehe Kapitel A.6*

7. Training und Tests LLM

- Trainings-CSV laden
- LLM laden
- LLM trainieren
- LLM testen
- Klassifizierungs-Ergebnisse ausgeben
- **Neuer Server: LLM-Container B.6 und C.3**
- *Jupyter-Notebooks siehe Kapitel A.7*

8. NN Auswahl

- Transformer Ergebnisse manuell vergleichen oder:
- KNN Ergebnisse laden und automatisiert vergleichen
- NN zum Mailserver kopieren für die Anwendung
- **Neuer Server: LLM-Container B.6, C.3; Alter Server B.5;**
- *Jupyter-Notebooks siehe Kapitel A.8*

4.5. Verwendete Software

4.5.1. Python

Python verwenden fast alle ML-Projekte. Leider verlangen unterschiedliche Software-Bibliotheken auch unterschiedliche Python-Versionen, die Python-Versionen ist aber auch vom verwendeten Betriebssysteme abhängig, darum werden hier unterschiedliche verwendet, siehe Tabellen: 4.2 und 4.3. Die verwendeten Python-Versionen sind: 3.8, 3.9, 3.10 und 3.11;

Abhängigkeiten

Leider ist die Python-Bibliotheken-Abhängigkeits-Hölle im LM-Bereich noch ausgeprägter als sonst. Es ist zu wenig, die Abhängigkeiten der Python-Bibliotheken untereinander zu berücksichtigen. Nur wenn 6

verschiedene Software-Bibliotheken zusammenpassen [129], funktioniert TensorFlow zuverlässig: TensorFlow, Python, Compiler, Build-Tools, cuDNN und CUDA.
Wenn eine GPU nicht nötig ist, wie es bei klassischen Modellen oder Transformer-Modellen im unteren dreistelligen MB-Bereich oft der Fall ist, kann auch die CPU-Version verwendet werden. Hier gibt es nur 4 Abhängigkeiten [129]: TensorFlow, Python, Compiler und Build-Tools.
Sollen aber aktuelle LLMs trainiert werden, ist eine GPU oft zwingend nötig. Dies macht schon alleine Sinn wegen der Rechenzeit, die damit eingespart wird. Der zwingende Grund ist, viele Bibliotheken setzen eine GPU voraus.

4.5.2. Jupyter

Jupyter-Notebook sind eine ausgezeichnete Möglichkeit, mit Python, grafischen Ausgaben und Diagrammen zu arbeiten. Daher wird es auch hier verwendet. Auch wenn die Endversion ein Python-Skript sein wird, macht es das Entwickeln wesentlich einfacher. Leider ist es von Python-Bibliotheken abhängig, die auch für das Trainieren des SPAM-Filters nötig sind, daher sind mehrere Versionen im Einsatz. Es wird in der Version 6.2.0 auf allen Debian-Systemen für das Übersetzen und für spaCy verwendet. Im LLM-Container wird die Version 7.2.2 verwendet.

4.5.3. PyTorch

Im Zuge dieser Arbeit sollte TensorFlow [130] durch PyTorch [54] ersetzt werden. Google hat es inzwischen durch den Nachfolger JAX ersetzt [52]. Aktuell werden 90% der neuen ML-Projekte in Torch, beziehungsweise der Python-Variante PyTorch, umgesetzt, siehe die Bücher [53] und [131]. In dem Container-LLM wird die torch in der Version 2.3.0 für Transformer-Modelle verwendet.
Der Aufwand, die alten Modelle und Notebooks auf PyTorch umzubauen, hat sich als zu hoch herausgestellt. Da viele verwendete Python-Bibliotheken nur mit dem alten TensorFlow zusammenarbeiten. Diese alte Version verhindert aber neue Bibliotheken. Daher wird TensorFlow in den Version 2.6.0 und 2.18.0 verwendet.

4.5.4. Argos-Translate

Beim Datensatz 1 hat sich bei den ersten Tests eine deutlich bessere Erkennungsrate als bei den anderen Datensätzen gezeigt. Beim Datensatz 1 ist der Großteil der Mails Englisch. Daher werden alle Mails auf Englisch übersetzt, um zu testen ob die Erkennungsrate dadurch besser wird. Fürs Übersetzen wird erst Trax in der Version 1.3.8 [132] und dann Argos-Translate in der Version 1.9.6 [133] getestet.

4.5.5. spaCy

spaCy wird für NLP-Aufgaben in der Version 3.8.0 verwendet. Der Stemmer-Algorithmus kommt von NLTK [32]. Es ist ein Python Code der sein Wissen aus WordNet [134] bezieht. NLTK steht für Natural Language Toolkit und wird als Vorgänger von spaCy betrachtet. Die verwendete Version ist 3.5-1.

4.5.6. Nvidia CUDA

Der Hersteller Nvidia mit seiner CUDA-Bibliothek dominiert hier den Markt. Man kann auch alternative Grafikkarten verwenden, allerdings unterstützen diese weniger Funktionen. Um ein LM-Projekt mit Python und GPU umzusetzen, muss eine vom Hersteller Nvidia unterstützte CUDA-Toolkit-, Python- und Nvidia-Treiber-Kombination gefunden werden, sonst führt das zu unvorhergesehenen Ergebnissen und Abstürzen. Auffällig ist hier, dass exakt die Python-Versionen ([135] und [136]) unterstützt werden, die bei Betriebssystem Ubuntu Linux in der LTS-Version ausgeliefert werden.

4.5.7. Nilsimsa

Textmenge zum Lernen: Bei klassischen Modellen braucht man eine sehr große Menge an Beispielen (Hunderttausende) zum Trainieren, und die Modelle müssen öfter als einmal mit denselben Tokens trainiert werden. Transformer-Modelle, zu denen auch BERT gehört, und alle LLMs kommen mit deutlich weniger Text aus. Es reichten schon tausende Beispiele für gute Ergebnisse.

Die klassischen Modelle werden mit einem Quartal an HAM und 6 Jahren an SPAM trainiert, dies sind hunderttausende Mails. Viele davon sind zwar ähnlich, aber selten exakt gleich. Nur die exakt gleichen können automatisch aussortiert werden. Das Ergebnis wird ungenauer, wenn fast gleiche Mails auch gelernt werden.

Mit dem Löschen der Mails mit gleichem Inhalt werden aus 3086344 die 2186066 vom Datensatz 6, Details in der Tabelle 4.1. Die vielen ähnlichen Mails sind zum großen Teil personalisierte Newsletter oder Roundtrip-Test-Mails zum Testen der Mailstrecke. Diese unterscheiden sich oft nur durch ein oder zwei Wörter. Meist nur durch die Mailadresse, eine fortlaufende Nummer oder Datum und Uhrzeit. Alleine rund 200 k Mails sind Roundtrip-Test-Mails. Daraus ergeben sich folgende 2 Aufgaben:

- Wie kann die Mailanzahl sinnvoll reduziert werden?

- Wie die unterschiedlichsten Mails auswählen?

Beide Aufgaben können mit Nilsimsa [21] gelöst werden. Es werden damit die Mails ausgewählt, die am unterschiedlichsten sind. So wie alle anderen Lösungen in dieser Arbeit arbeitet Nilsimsa nur mit dem Text aus dem Mail-Body.

4.6. Aufbau der Testumgebung

Da der Autor GNU Debian Linux seit über 25 Jahren verwendet, werden die ersten Berechnungen auf einem Laptop mit der CPU auf Debian gemacht. War eine GPU notwendig, wird ein Desktop, später zwei Server verwendet. Die verwendeten Betriebssysteme waren hier Debian, Ubuntu und RedHat. Ubuntu wird vom Hersteller Nvidia und von vielen Data-Scientists verwendet, daher ist Ubuntu zu bevorzugen. Es gibt hier die größte Auswahl an Bibliotheken zum Thema NLP. Die offiziellen Nvidia Container und Nvidia Entwicklungsrechner wie die Jetson-Serie basieren ebenfalls auf Ubuntu. Ein Übersicht bietet diese Tabelle 4.3.

Leider unterscheiden sich bei Python die Abhängigkeiten der Versionen je nach Bibliothek stark. Es führt oft dazu, dass Bibliothek A und B NICHT nebeneinander installiert werden können, da sie eine unterschiedliche, inkompatible Version von der Bibliothek C benötigen. Die klassische Lösung ist hier, zwei virtuelle Umgebungen für Python zu erstellen und dort die nötigen Bibliotheken getrennt voneinander zu installieren. Dies ist keine Lösung für viele ML-Bibliotheken, da diese auch noch eine bestimmte CUDA-Version, die vom Nvidia-GPU-Treiber abhängt, benötigen. Daher ist das klassische virtuelle Environment von Python unzureichend. Daher ist es nötig, entweder ein eigenständiges Betriebssystem in einer Virtualisierungsumgebung oder eine Container-Lösung zu verwenden.

Es wird hier mit Containern gearbeitet. Es wird die Podman-Lösung verwendet. Leider gibt es die Nvidia-GPU-Treiber in drei Versionen: einer für Consumer, einer für Enterprise und einer für den Jetson. Die sind leider inkompatibel und daher muss auch der Container angepasst werden.
Am Jetson AGX Orin wird mit Jetson-Containern gearbeitet, da diese von Nvidia am besten unterstützt werden. Diese Lösung basiert auf Docker, der LLaMA-Factory-Container läuft darauf.
Es wird für die verschiedenen Aufgaben insgesamt mit sieben Containern in unterschiedlichen Versionen gearbeitet, auf 5 verschiedenen Rechnern, siehe Tabelle 4.5.

4.7. Containerarten

Es werden je nach Aufgabe unterschiedliche Container in unterschiedlicher Konfiguration verwendet, Details siehe Anhang C und B. Es wird Podman in der Version 3.0.1 verwendet, auf dem Laptop und Desktop. Die Version 4.9.4 auf dem neuen Server, siehe Tabelle 4.5. Am Jetson wird Docker in der Version 5.27.4 verwendet , siehe Tabelle 4.4.
Es werden 5 verschiedene Jupyter-Notebooks in unterschiedlichsten Konfigurationen auf verschiedenen Containern verwendet, wegen der unterschiedlichen Abhängigkeiten.
Berechnungsumgebungen:

- **Klassik:** die klassischen Modelle am Laptop, Desktop oder am alten Server. Verwendete Software in Abschnitt B.1, B.3 und B.5
- **Jupyter-Notebook:** Nilsimsa Auswahl der Mails und FastText für das Erkennen der Sprache am neuen Server im Container in Abschnitt C.1. Verwendete Software in Abschnitt B.2
- **Translate:** Textvorverarbeitung mit spaCy und Übersetzung am neuen Server im Container in Abschnitt C.2. Verwendete Software in Abschnitt B.4
- **LLM:** Trainieren von Transformer-Modellen am Neuen Server im Container in Abschnitt C.3, Software B.6
- **LLaMA Factory:** LLaMA Factory am Jetson AGX Orin im Container in Abschnitt C.4
- **Whisper:** Transkribieren von gesprochenen Teilen der Arbeit am Jetson AGX Orin im Container: whisper von https://github.com/dusty-nv/jetson-containers

4.8. Hardware

Details zu den verwendeten Hardwareplattformen sind in folgender Tabelle zu finden 4.2. Erste Tests werden mit wenig Mails und kleinen NN auf alter Hardware durchgeführt. Nur wenn es funktioniert hat und die Ergebnisse vielversprechend waren, werden die Tests ausgeweitet. Erste Tests werden am Laptop, der keine Nvidia-GPU hat, oder am Desktop, der eine Nvidia RTX 4070 besitzt, gemacht, abhängig davon, ob eine GPU nötig war.
Einfache Modelle funktionieren auch auf CPU, daher ist ein alter Laptop für Tests auch brauchbar, es ist aber keine Option mehr bei LLMs, das ist unabhängig von der Größe der Modelle, es gibt hier auch kleinere, aber manche Software-Bibliotheken setzen eine GPU voraus.
Jetson AGX Orin ist ein Entwicklungssystem von Nvidia. Er biete bis zu 64 GiB VRAM, wenn auch nur rund ein 1/10 der Rechenleistung einer aktuellen Grafikkarte. Grafikkarten mit so viel VRAM sind unbe-

zahlbar (fünfstellige Euro-Beträge). Am Jetson werden die Tests gemacht, die viel VRAM benötigen, wie das Trainieren von LLMs über LLaMA-Factory.

Die Anzahl der Token pro Mail (=Kontextlänge, siehe Abschnitt 2.2.4) wird auf die ersten 512 begrenzt. Dies wirkt sich auch positiv auf die erforderliche Rechenleistung und den Speicherverbrauch aus.

Am alten Server werden die klassischen Modelle getestet, da diese am neuen Server mit der neuen CUDA-Treiberversion inkompatibel sind.

Die Tests, die viel Rechenleistung brauchten, werden auf dem neuen Server durchgeführt. Der hat zwei Nvidia L40, diese bieten 2 x 44 GB Speicher und mehr Rechenleistung.

Plattform -typ	**Kerne Anz.**	**CPU GPU**	**Takt GHz**	**Größe nm**	**el.Leist. W**	**VRAM GB**	**CUDA-kerne**
Laptop	4	Intel Core i7-8550U	1,8	14	25		
Desktop	8	AMD Ryzen 7 5700X	4,6	7	65		
	1	Nvidia GeForce RTX4070	2,48	5	200	12	5888
Jetson AGX	12	Arm Cortex-A78AE v8.2	1,3	7	50		
-Orin	1	Nvidia Ampere	1,3	7	50	64	2048
Server	2 x 6	Intel Core i7-8700K	3,7	14	2 x 95		
-Alt	2	Nvidia GeForce RTX2070	1,62	12	2 x 385	8	256
Server	2 x 8	Intel Xeon Gold 5415+	4,1	10	2 x 150		
-Neu	2	Nvidia L40	2,52	4	2 x 300	44	9334

Tabelle 4.2.: Verwendete Hardware: Typ der Plattform, Anz. der Kerne, 1. Zeile verwendete CPU, 2. Zeile verwendete GPU, Strukturbreite in nm, Leistungsaufnahme, VRAM und CUDA-Kerne der GPU.

4.9. Vorverarbeitung

Hier werden die Ausgangsdaten, die archivierten Mails, in die richtige Form für das weitere Verarbeiten gebracht. Das meiste wird schon beim Archivieren selbst gemacht. Die ersten großen LLMs haben ja nur auf Datenmenge gesetzt, das hat zwar funktioniert, aber es gibt auch andere Erfolgsparameter. Die Qualität der Daten ist ebenso wichtig, siehe Kapitel 2.5.8. Daher ist der wichtigste Schritt, die Daten zu bereinigen.

Plattform -typ	OS Name	Python Version	CUDA Version
Laptop	Debian GNU/Linux 11.11	3.9	-
Desktop	Debian GNU/Linux 12.8	3.11	12.6
Jetson v. Upgrade	Ubuntu 20.04.6 LTS	3.8	11.4
Jetson AGX Orin	Ubuntu 22.04.4 LTS	3.10	12.6
Server Alt	Debian GNU/Linux 11.11	3.9	12.4
Server Neu	Red Hat Enterprise Linux 9.5	3.9	12.7

Tabelle 4.3.: Verwendete Hardware: OS, Python-Version und CUDA-Treiber-Version

4.9.1. Text reinigen

Das Bereinigen der Daten ist ein wichtiger Teil des Ablaufs [100]. Hierfür wird die spaCy-Bibliothek verwendet. Es ist eine beliebte Python-ML-Bibliothek, die auf TensorFlow, aber auch auf Torch aufsetzen kann. Die Aufgabe von spaCy ist, das Arbeiten mit Text zu vereinfachen. Eigentlich sollte das meiste schon beim Archivieren der Mails passiert sein, aber es kommt immer wieder vor, dass hier noch HTML-Fragmente zu finden sind. Auch soll versucht werden, ob es hilft, Stoppwörter und andere unwichtige Teile des Mailtexts zu entfernen.

spaCy bietet eine Reihe von praktische Funktionen, die getestet werden sollen:

- Erkennung von Satzende, um die Mail aufzuteilen
- Löschen von Daten: Mailadressen, URLs, Zahlen, Leerzeichen, Satzzeichen, Stoppwörter
- Erkennen von Wortarten und Wichtigkeit des einzelnen Worts im Satz

4.9.2. Textvorbereitung

Die meisten SPAM-Filter arbeiten ausgezeichnet bei englischsprachigen Mails, aber deutlich schlechter bei anderen Sprachen. Die Vermutung war, dass es einfach eine Frage der Quantität ist, da die meisten Mails englischsprachig sind.

Das Ziel der ursprünglichen Arbeit [8] war, mit selbst erstellten klassischen NN dieses Problem zu beheben. Es werden 3 neue leere Modelle getrennt trainiert. Ein NN für Deutsch, eines für Englisch und eines für alle anderen Sprachen. Dieser Ansatz half, aber die Ergebnisse für Englisch sind trotzdem immer noch am besten.

Plattform -typ	**CUDA-Treiber -Paket**	**Toolkit -Name**
Laptop	-	-
Desktop	nvidia-driver/stable-updates,now 525.147.05-7 ~deb12u1 amd64	Nvidia-cuda-toolkit/ stable,now 11.8.89 ~11.8.0-5 ~deb12u1 amd64
Jetson v. Upgrade	cuda_11.4.r11.4/ compiler.31964100_0	Nvidia-cuda-toolkit/ jammy 11.5.1-1ubuntu1 arm64
Jetson AGX Orin	cuda_12.6.r12.6/ compiler.34714021_0	Nvidia-cuda-toolkit/ oldstable 11.2.2-3+deb11u3 amd64
Server Alt	cuda-repo-debian12-12-4-local/ now 12.4.1-550.54.15-1 amd64	
Server Neu	Nvidia-driver-cuda-565.57.01-1. el9.x86_64	Nvidia-container-toolkit -1.17.3-1.x86_64

Tabelle 4.4.: Verwendete Hardware: GPU Treiber und Toolkit-Version

Eine andere Vermutung ist, es liegt an der Grammatik oder am Wortaufbau. Da es im Deutschen viel mehr verschiedene Wortendungen gibt als im Englischen oder das Deutsche Wörter einfach länger sind, z.B. Donaudampfschifffahrtskapitän. Der Wortaufbau soll daher vereinfacht werden. Es gibt hierfür die beiden Verfahren Stemming und Lemma, das sind Algorithmen, die den Wortstamm [137] finden.
Lemmatisierung ist aufwendiger und liefert auch für unterschiedliche Wortstämme, die dieselbe Bedeutung haben, nur einen einmaligen Wortstamm zurück. Auch hier bietet spaCy viele Möglichkeiten. Es gibt bei Englisch mehrere Algorithmen zur Auswahl, für viele andere Sprachen nur einen oder gar keinen.

NER

Bei NER wird die Wortkategorie erkannt. Es wird bei spaCy automatisch bei der Satzanalyse, die immer am Anfang eines Workflows steht, gleich mit durchgeführt. Das gilt auch für POS und DEP.

POS

spaCy erkennt die Wortart und speichert sie als POS (Part of Speech). Wenn das mitberücksichtigt wird, bleibt der Unterschied zwischen einem Hauptwort und einem Zeitwort trotz Lemmatisierung erhalten. Auch ist damit das Problem von Wörtern mit selber Schreibweise, aber mit unterschiedlicher Bedeutung gelöst,

Plattform	OS Isolierung	Paketname und Version
Laptop	Container	podman/oldstable,now 3.0.1+dfsg1-3+deb11u5 amd64
Laptop	Virtualisierung	VirtualBox 6.1.40_Debian r154048
Desktop	Container	podman/stable,now 4.3.1+ds1-8+deb12u1 amd64
Jetson v. Upgrade	Container	docker-ce ~ubuntu.20.04 ~jammy arm64
Jetson AGX Orin	Container	docker-ce/jammy,now 5:27.4.1-1 ~ubuntu.22.04j̃ammy arm64
Server Alt	-	-
Server Neu	Container	podman-5.2.2-11.el9_5.x86_64

Tabelle 4.5.: Verwendete Hardware: Container oder Virtualisierung Version

da jetzt Zusatzinformation zur Verfügung steht, die bei der richtigen Zuordnung hilft.

DEP

Mit dem DEP-Token (syntactic DEPendency) wird die Beziehung der Wörter untereinander analysiert. POS zeigt an, wie ein Wort auf ein anderes verweist. Er hilft also, das wichtige Wort im Satz zu finden. Damit wird die Intention klarer.

LPD

Die drei Werte Lemma, POS und DEP (LPD) werden zu einem neuen Wort zusammengesetzt, da die Analyse im ML ja auf Wortebene basiert. Damit kann dann das NN gefüttert werden und es hat damit auch noch die Zusatzinformation gleich beim Wort gespeichert. Das geht nur bei Modellen, die kein vorgefertigtes Wörterbuch haben, da es neue Kunstwörter sind. Es funktioniert aber nur bei den klassischen Modellen, die von Grund auf selbst trainiert werden.

Stoppwortentfernung

Die Stoppwörter werden über eine spaCy-Funktion entfernt. Sie werden im spaCy-Workflow automatisch erkannt.

Verteilung der Sprachen

Da bei unterschiedlichen Sprachen oft unterschiedliche Verfahren oder NN nötig sind, hier eine Übersicht der Sprachverteilung für den größten verwendeten Datensatz 6, siehe Tabelle 4.6.

Sprache	Mailanz.	% von Mails
Deutsch	1557371	71,24
Englisch	569592	26,06
Russisch	29256	1,34
Slowakisch	8590	0,39
Tschechisch	4220	0,19
Französisch	3412	0,16
Italienisch	3296	0,15
Ungarisch	2876	0,13
Spanisch	1884	0,09
Niederländisch	649	0,03
Sonstige	4916	0,22

Tabelle 4.6.: Sprachverteilung der Mails im Datensatz 6, Anzahl und Prozent

Die Mails sind meist Deutsch, dann kommt weit abgeschlagen Englisch und nochmal weit dahinter alle anderen Sprachen. Nicht überraschend ist hier, dass die Sprachen der österreichischen Nachbarländer hier vorne liegen, neben den internationalen Verkehrssprachen wie Englisch und Französisch. Russisch ist seit dem Start des russischen Angriffskriegs gegen die Ukraine im Jahr 2022 stark verbreitet, allerdings hauptsächlich als SPAM. Mit Unterstützung von Deutsch und Englisch sind 97,30% der Mails abgedeckt. Es bleiben nur 2,70% für andere Sprachen übrig.

Bei den kleineren Datensätzen ist die Verteilung ähnlich, ausgenommen der Enron-Datensatz, dort ist fast alles Englisch, siehe Abschnitt 4.3.2.

4.9.3. Übersetzung

Da die Kategorisierung von englischen Mails besser funktioniert als für andere Sprachen, sollen die Mails vor der Verarbeitung ins Englische Übersetzt werden, siehe Tabelle 4.7.

Damit wird die Sprache verwendet, die die besten Ergebnisse liefert, und jeder weitere Verarbeitungsschritt wird damit unabhängig von der Sprache. Auch bietet das den Vorteil des kleineren Wörterbuchs, siehe Abschnitt 2.2.2. Alle ML-Modelle arbeiten nie mit den eigentlichen Wörtern selbst, sondern über deren Nummern aus dem Wörterbüchern. Die Größe dieser Wörterbücher ist begrenzt. Wird hier nur eine Sprache berücksichtigt, ist die Anzahl der möglichen Wörter wesentlich kleiner und es werden mehrere Wörter einer

Mail berücksichtigt.

	de	en	so
kurz	6,4	1,9	4,5
mittel	7,5	2,4	1,7
lang	15,8	8,6	14,0

Tabelle 4.7.: Fehler der SPAM-Klassifizierung in %, abhängig von der Sprache und Länge der Nachrichten, Tabelle wurde von S.30 dieser Arbeit [8] übernommen

4.10. Test der Modelle

4.10.1. Auswahl der Modelle

Es werden hier in erster Linie Modelle angeschaut, die auch für den Chatbot Ollama [138] verfügbar sind. Dieser gibt eine gute Übersicht über aktuelle beliebte Modelle, die auch viel verwendet werden. Danach werden die Modelle näher angeschaut, die auf der Webseite von HuggingFace zu finden waren. HuggingFace (HF) [139] ist derzeit eine sehr beliebte Website/Repository für ML/NLP/LLM-Technologien.
Da HF etwa 85k Modelle zur Auswahl anbietet, muss die Auswahl/Suche von Modellen für unseren Anwendungsfall eingeschränkt werden. Es werden nur die Modelle in den folgenden beiden Unterkategorien von „Natural Language Processing“ angesehen: „Text Classification“ und „Zero-Shot Classification“. Hier wird nach „Trending“ und „Most Downloads“ sortiert,[140]. Modelle direkt vom Hersteller werden gegenüber angepassten Versionen bevorzugt. Dieser Auswahlprozess führte zu den verschiedenen Modellen für unsere Bewertung, siehe Tabelle 4.8.

4.10.2. LLM-Chat-Bot

Die aktuellen LLM-Chat-Bots sind inzwischen sehr gut, aber können sie auch für die Erkennung von SPAM-Mails verwendet werden?
Getestet wird erst manuell, später solle es dann über API automatisiert werden. Der Online-Dienst GPT 3.5 wird zwar getestet, aber da Mails oft persönliche oder vertrauliche Daten enthalten, wird auch eine Lösung gesucht, die sich DSGVO-konform umsetzen lässt. Das ist mit einem lokalen LLM wie Mixtral 8 x 7B möglich [120].

4.10.3. LlaMA-Factory

Das Trainieren von Transformer-Modellen wird im ersten Schritt mit LLaMA-Factory versucht, da es hier eine einfache grafische Oberfläche mit vielen Möglichkeiten gibt, siehe Abschnitt 2.5.12. Später wird dann mit Jupyter-Notebooks gearbeitet.

4.10.4. Änderungen an den klassischen Modellen

Damit die klassischen Modelle vergleichbarer sind, werden die Wörterbuchgröße und die Kontextlänge, also die berücksichtigte Anzahl an Wörtern/Token, angepasst.

Kontextlänge

Die Kontextlänge wird von den alten 300 auf 512 erhöht, da es eine übliche Größe für BERT-Modelle ist. Getestet wird auch 128, da es die Breite des ersten BERT-Modells war, siehe Tabelle 2.2.4.

Wörterbuchgröße

Um die klassischen mit den neuen NN besser vergleichbar zu machen, wird die Wörterbuchgröße angepasst, siehe Tabelle 4.9.

Epochen

Bei klassischen NN braucht es viele Epochen, um ein Modell von einer gewissen Qualität zu bekommen. Bei BERT und LLMs reicht es oft aus, nur eine Epoche zu trainieren. Daher ist der gesamte Rechenaufwand bei den klassischen NN und BERT in derselben Größenordnung.

4.10.5. BERT und LLM

Es werden verschiedene BERT- und LLM-Modelle mit denselben Mails wie die klassischen Modelle trainiert und anschließend wird das erstellte Modell mit einem kleinen Datensatz getestet, siehe [159] und [120]. Es wird hier ebenfalls ein unterschiedlicher Wert für die Kontextlänge getestet, um zu sehen, wie sich das auf das Trainieren und die Genauigkeit auswirkt. Wie groß genau die Gleitkomma-Variable eines einzelnen Tensors ist, kann variiert werden, da gibt es Werte zwischen 64 und 4 Bit, das wird getestet. Ein wichtiger Faktor ist auch, mit wie vielen Beispieldaten ein NN trainiert werden muss, damit es gute Ergebnisse liefert. Daher wird auch das analysiert.

4.11. Bedarf an elektrischer Energie

Wichtig ist, wie viel Energie wird verbraucht. Die SI-Einheit hierfür ist Wh. Wh steht für die elektrische Leistung pro Stunde, meist wird bei der Stromrechnung kWh verwendet. Der genaue CO_2-Ausstoß wird ignoriert, dieser hängt ja auch vom Wirkungsgrad und von der Art der Energieerzeugung ab [160].

4.12. Rechtliche Aspekte

Es werden die DSGVO und der neue AI-Act der EU analysiert, inwiefern sie für das aktuelle Projekt relevant sind.

[tb] **Name**	**Type**	**Paper**	**Quelle/HuggingFace-Name**
Dense	Dense	[141]	[56]
LSTM	LSTM	[57]	[56]
Bi-LSTM	Bi-LSTM	[142]	[56]
CNN	CNN	[143]	[144]
CNN3	Multi-CNN	[145]	[62]
M-CNN5	Multi-CNN	[145]	[62]
Bart	BERT	[146]	facebook/bart-large-mnli
Bge-reranker	BERT	[147]	BAAI/bge-reranker-v2-m3
LL32 1B	Transformer	[69]	meta-llama/Llama-3.2-1B
LL32 3B	Transformer	[69]	meta-llama/Llama-3.2-3B
Phi-1.5	Transformer	[148]	microsoft/phi-1_5
Phi-3	Transformer	[149]	microsoft/Phi-3-mini-4k-instruct
Phi-4	Transformer	[150]	microsoft/phi-4
Teuken	Transformer	[70]	openGPT-X/Teuken-7B-instruct-research-v0.4
MistralNemo	Transformer		mistralai/Mistral-Nemo-Instruct-2407
SmolLM2	Transformer	[151]	HuggingFaceTB/SmolLM2-1.7B-Instruct
DeepSeek 14B	Transformer	[152]	deepseek-ai/DeepSeek-R1-Distill-Qwen-14B
Mistral-7B	Transformer	[153]	mistralai/Mistral-7B-Instruct-v0.3
Mixtral 8 x 7B	Transformer	[154]	mistralai/Mixtral-8x7B-Instruct-v0.1
GPT 2	Transformer	[155]	openai-community/gpt2
GPT 3.5	Transformer		kein öffentliches Modell
FastText	CNN	[36]	facebook/fasttext-language-identification
Argos-Translate	Transformer	[156]	[157]
en_core_web_sm	CNN	[158]	spaCy/en_core_web_sm
de_core_news_sm	CNN	[158]	spaCy/de_core_news_sm
xx_ent_wiki_sm	CNN	[158]	spaCy/xx_ent_wiki_sm
en_core_web_trf	BERT	[158]	spaCy/en_core_web_trf
de_dep_news_trf	BERT	[158]	spaCy/de_dep_news_trf

Tabelle 4.8.: Übersicht über die verwendeten NN: Name, Type, Paper, Quelle oder HuggingFace-Name

Name	Größe in GB	Wörter- -buchgr.	Kontext- länge max.	Knowledge-Cut- Monat	-off-date Jahr
Dense	0,012	32768	512	März	2024
LSTM	0,012	32768	512	März	2024
Bi-LSTM	0,013	32768	512	März	2024
CNN	0,016	32768	512	März	2024
CNN3	0,016	32768	512	März	2024
M-CNN5	0,049	32768	512	März	2024
Bart	1,5	50256	1024	vor	2019
Bge-reranker	2,3	25000	8194	vor Februar	2024
LL32 1B	2,4	128256	131072	Dezember	2023
LL32 3B	6,5	128256	131072	Dezember	2023
Phi-1.5	2,8	50256	2048		2023
Phi-3	29,3	32046	4096	Oktober	2023
Phi-4	7,6	100352	16384	Juni	2024
Teuken	14,6	250680	4096	Dezember	2023
MistralNemo	24,5	131072	1E+030	Juli	2024
SmolLM2	3,3	49152	8192	vor Oktober	2024
DeepSeek 14B	29,6	131072	16384	vor Dezember	2024
Mistral-7B	14,5	32768	256	September	2022
Mixtral 8 x 7B	96,8	32000	65536	Dezember	2023
GPT 2	0,498	50256	1024	November	2019
GPT 3.5	350			September	2021
FastText	0,126			vor	2016
Argos-Translate Min.	0,067			vor März	2024
Argos-Translate Max.	0,156			vor März	2024
en_core_web_sm	0,012			vor Septmber	2024
de_core_news_sm	0,013			vor Septmber	2024
xx_ent_wiki_sm	0,01			vor Septmber	2024
en_core_web_trf	0,436	50256	768	vor Septmber	2024
de_dep_news_trf	0,391	30000	768	vor Septmber	2024

Tabelle 4.9.: Eigenschaften der verwendeten NN: Größe des Modells im Speicher, Wörterbuchgröße, maximale Kontextlänge, Knowledge-Cut-off-Date (wenn keines bekannt ist, wird **vor** dem Veröffentlichungsdatum angenommen)

5. Herangehensweise

5.1. Szenario

In dieser Arbeit soll untersucht werden, ob ein bestehendes System zur SPAM-Filterung verbessert werden kann, indem neue Techniken integriert werden, darunter auch LLMs.

Der bestehende Aufbau verwendet für den SPAM-Filter unter anderem klassische NN(siehe Abschnitt 2.4), die auf mehreren CPU-Kernen laufen. Das funktioniert einigermaßen, aber die absolute Anzahl der SPAM-Mails hat sich im letzten Jahr verdreifacht das gilt auch für die Anzahl der False-Negatives, die sind im selben Verhältnis gestiegen. Daher wäre eine Verbesserung bei der SPAM-Erkennungsrate wünschenswert.

5.2. Testablauf

Als Erstes wird die Vorverarbeitung der Daten verbessert. Anschließend werden neben den klassischen NN auch neue BERT-Modelle und LLMs getestet.

5.3. Vorverarbeitung

Es werden hier die erkannte Sprache der Mails und die Wahrscheinlichkeit dafür gespeichert. Die Mails werden nach Länge in drei Gruppen aufgeteilt. Dann wird der Text gereinigt.

FastText

Für viele Operationen ist es wichtig, die Sprache zu erkennen. Das wird mit FastText [36] durchgeführt.

Text Bereinigung

Da BeautifulSoup einige HTML-Fragment übrig gelassen hat, werden verbleibende HTML-Schlagwörter wie „mailto“ gelöscht, damit die Satzerkennung funktioniert. spaCy hat Probleme mit Wörtern wenn sie durch einen Schreibfehler mit einem Punkt oder einer Zahl beginnen. Daher werden um Zahlen/Punkte Leerzeichen eingefügt, da der Tokenizer überzählige Leerzeichen ohnehin ignoriert.
Es werden spaCy-Modelle für verschiedene Sprachen getestet. Es gibt die spaCy-Transformer-Modelle nur für manche Sprachen der vorliegenden Mails und diese sprachspezifischen Modelle brauchen wesentlich mehr Rechenleistung als die Modelle, die keine bestimmte Sprache unterstützen, siehe Tabelle 6.22. Daher werden weitere Tests mit dem sprachunabhängigen kleinen Modell durchgeführt.

Text kürzen

Für den Vergleich sollte die Anzahl der verwendeten Wörter gleich sein. Daher wird die Anzahl der Wörter pro Mail auf die ersten 512–554 begrenzt. Da die alten Modelle nur mit 300 Wörtern arbeiteten und viele BERT-Modelle nur 512 Token für die Kontextlänge unterstützen. Dies wirkt sich positiv auf die nötige Rechenleistung und den Speicherverbrauch aus.
Die durchschnittliche Maillänge inkl. Leerzeichen und Satzzeichen beträgt beim größten Datensatz 6 2462 Zeichen. Über die durchschnittliche Wortlänge von 6,6 Zeichen (siehe [161]) plus ein Leerzeichen kommen 324 Wörter heraus. In die andere Richtung gerechnet bedeuten 512 Wörter eine Mailtextlänge von 3891 Zeichen. Damit werden mehr als 80 % der Mails abgedeckt. In der Vorverarbeitung direkt nach der Archivierung werden Mails, die länger als 31999 Zeichen sind, auf diese Länge begrenzt. Hier die Längenverteilung der Mails, siehe Tabelle 5.1.

Perzentile in % →	**min.**	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**90**	**max.**
Zeichenanzahl	7	272	495	787	1227	1703	2274	2875	3395	5188	31998

Tabelle 5.1.: Längenverteilung der Mails im Datensatz 6

Die variable Grenze von 512–554 wird gewählt, um keine Sätze abzuschneiden, da dies sich ungünstig auf die Übersetzungsqualität auswirkt, siehe Abschnitt 2.2.4.
Es gibt leider Mails, die nur aus einer Handvoll Sonderzeichen und keinem einzigen Wort bestehen. Diese führen bei der Übersetzung zum Absturz. Diese werden gefiltert, indem Mails mit weniger als fünf Zeichen gelöscht werden. Bei dem Datensatz 6 mit dem Grenzwert 46, der aus 103802 Mails besteht, sind davon 16 Mails betroffen, siehe Tabelle 6.10.

Stemmer

Die mit großem Abstand am weitesten verbreiteten Sprachen im Testset sind Deutsch und Englisch, gefolgt von europäischen Sprachen, siehe Tabelle 4.6. Die meiste Auswahl an Stemmern gibt es für Englisch. Hier wird der am weitesten verbreitete und immer noch weiterentwickelte Porter-Stemmer-Algorithmus [162] verwendet. Bei Deutsch gibt es eigentlich nur einen Stemmer, der gut funktioniert, es wird der Algorithmus Cistem [163] verwendet. Für alle anderen Sprachen wird der Snowball-Stemmer [164] verwendet, da er sechzehn weit verbreitete europäische Sprachen unterstützt. Alle Stemmer-Algorithmen verwenden die Python-Bibliothek NLTK. Für die ersten Tests wird der Datensatz 3 verwendet, siehe Tabelle 4.1.

Lemmatisierung

Lemmatisierung wird als Nächstes getestet. Es braucht zwar mehr Rechenleistung als der Stemmer-Algorithmus, liefert aber auch bessere Ergebnisse.
Für den Lemmatisierungs-Algorithmus wird spaCy verwendet. Dieser wird ohnehin automatisch mit angewendet, sobald ein Text mit spaCy analysiert wird, siehe Tabelle 6.22. Es werden hier 2 größere Transformer-Modelle für Deutsch und Englisch getestet.
Für die meisten Sprachen gibt es aber gar keine Transformer-Modelle. Daher wird auch ein kleines sprachunabhängiges Modell [165] verwendet. Die fürs Training verwendeten Daten stammen von Wikipedia und es ist für die CPU optimiert. Diese kleinen Modelle werden auch für Deutsch und Englisch getestet, denn die Transformer-Modelle brauchen viel Rechenleistung und Speicher.

NER POS DEP und LDP

Als Erstes wird die Sprache mittels FastText erkannt und dann auf 3 Sprachgruppen aufgeteilt: Deutsch, Englisch und Sonstige, siehe Abschnitt 2.2.3. Es wird jeweils versucht, die Ergebnisse des SPAM-Filters über das Hinzufügen von Zusatzinformationen mit spaCy zu verbessern. Dazu wird erst die jeweilige Zusatzinformation für jedes Wort zum eigentlichen Text hinzugefügt und dann die 6 eigenen klassischen Modelle trainiert und anschließend getestet, siehe Tabelle 5.2.
Die Zusatzinformationen werden über folgende Arten erlangt:

- Eigennamenserkennung (NER)
- Wortart (POS)
- Beziehung der Wörter zueinander (DEP)
- LPD besteht aus: Lemma, POS und DEP

Stoppwortentfernung

Es wird versucht, den Text zu vereinfachen, indem die Stoppwörter gelöscht werden, bevor die NN trainiert und getestet werden.

5.3.1. Übersetzung

Die Kategorisierung von Mails nach SPAM funktioniert im Englischen besser als in anderen Sprachen, siehe auch [34] und [33]. Daher besteht eine Strategie darin, die Mails zunächst automatisch ins Englische zu übersetzen. Wenn nur Texte in einer Sprache verwendet werden, verringert sich die Anzahl der verschiedenen Wörter, das Wörterbuch ist damit genauer und deckt den Text besser ab, siehe Abschnitt 2.3.
Im größten Datensatz 6 werden mit FastText 95 verschiedene Sprachen gefunden, siehe Tabelle 4.6. Zur Verteilung der Sprachen: 71,24% der Mails sind Deutsch, dann kommt Englisch mit 26,06% und mit großem Abstand dahinter Russisch mit 1,34%. Alle anderen Sprachen zusammen machen nur 1,37% aller Mails aus.

5.3.2. FastText

Als Quellsprache fürs Übersetzen wird die von FastText erkannte Sprache herangezogen. Englische Mails liegen bereits in der richtigen Sprache vor. Bei deutschsprachigen Mails funktioniert die Übersetzung ausgezeichnet. Daher werden diese für die weiteren Tests aus dem Datensatz 6, der aus 2,2 Millionen Mails besteht, gefiltert. Mit den übrigen Mails wird genauer untersucht, ob alle Sprachen übersetzt werden können.

5.3.3. Argos-Translate

Argos-Translate hat schon beim ersten Versuch ausgezeichnete Ergebnisse geliefert und steht für viele Sprachen zur Verfügung. Es stehen für 43 Sprachen direkte Übersetzungsmodelle zur Verfügung. Anschließend werden dann noch 13 ähnliche Sprachen einem vorhandenen Modell zugewiesen.
Es ist Open Source und verwendet als Basis OpenNMT/CTranslate2. Dies wird seit 2016 von Harvard NLP entwickelt. Die Modelle verwenden Transformer [166]. Es liefert bessere Ergebnisse [40] als LLMs.
Es verwendet kleine ML-Modelle. Für jede Quell- und Zielsprache-Kombination braucht es ein eigenes Modell. Die Modelle sind zwischen 60 und 160MB groß. Für die andere Richtung braucht es ein eigenes Modell. Deutsch ist zwar die am weitesten verbreitete Sprache in einem österreichischen Unternehmen und daher wären nur wenige Mails zu übersetzen, aber ist da die SPAM-Erkennungsrate bei Englisch besser. Auch gibt es für Deutsch als Zielsprache weniger Modelle als für Englisch, hier müsste ein Übersetzungsumweg über andere Modelle gemacht werden. Hier ein Beispiel: Ein Text wird erst von Albanisch auf

Englisch und dann von Englisch auf Deutsch übersetzt. Dieser Umweg schadet der Qualität und verdoppelt die nötige Rechenleistung. Daher werden alle Mails nur auf Englisch übersetzt.

5.3.4. Übersetzung anpassen

Gibt es für manche Sprachen kein Modell, aber eines für eine verwandte Sprache, dann wird diese ausgewählt. Z.B. für Afrikaans wird das ähnliche Niederländisch dafür das Niederländisch-zu-Englisch-Modell verwendet. Im Kurdischen gibt es viele persische Lehnwörter, daher wird Persisch verwendet. In Indien gibt es viele Sprachen, die von kleinen Gruppen gesprochen werden. Hier wird Hindi ausgewählt. Damit wird ein Großteil der Mails fast vollständig übersetzt und kann damit für die Mail-Klassifizierung verwendet werden. Es gibt 43 Sprachen, die direkt ins Englische übersetzt werden können. Mit den ähnlichen Sprachen sind dann 56 abgedeckt.

Wenn Mails übersetzt werden, behebt das auch das Problem mit der Rechtschreibung, denn es werden damit alle Rechtschreibfehler automatisch behoben. Bei der Kontrolle der Ergebnisse der übersetzten Mails fällt auf, dass manchmal ganze Absätze trotzdem die Originalsprache haben. Das hat bis zu zwei Drittel einer Mail betroffen. Der Grund ist, dass in diesen Mails mehrere verschiedene Sprachen verwendet werden. Daher wird die Spracherkennung und die folgende Übersetzung von pro Mail auf pro Satz umgestellt. Die statistischen Werte, welcher Sprache eine Mail zugeordnet wird bleiben unverändert.
Als Erstes wird es mit dem Maildatensatz 3, der aus 25 k Mails besteht, im kleinen Rahmen getestet. Dann werden die Tests bis zum großen Datensatz 6 ausgeweitet, der aus 2,2 Millionen Mails besteht, siehe Tabelle 4.1. Für die größeren Tests wird auf dem neuen Server mit einer L40-GPU gerechnet. Da hat sich aber schnell gezeigt, alle Mails zu übersetzen dauert rund einen Monat. Daher wird eine Lösung gesucht, um das zu beschleunigen, ohne dabei wichtige Informationen zu verlieren. Die Lösung war Nilsimsa, siehe Abschnitt 5.3.5.

5.3.5. Nilsimsa berechnen

Der Gesamtvergleich für alle Mails braucht Jahre an Zeit. Die Lösung ist, die Anzahl der Vergleiche zu reduzieren, indem mehrere kleine Vergleichsgruppen erstellt werden. Schon bei der Vorverarbeitung werden die Sprache der Mails und deren Länge gespeichert. Auch ist im Datensatz gespeichert, ob eine Mail HAM oder SPAM ist. Dies wird jetzt verwendet, um die Anzahl der Vergleiche zu reduzieren.
Die Mails werden anhand dieser Kriterien in Gruppen aufgeteilt:

- Ist das Mail SPAM oder HAM?

- In welche der drei Mailllängengruppen gehört es?
- In welcher Sprache wurde das Mail am wahrscheinlichsten geschrieben?
- Wenn das Mail Deutsch oder Englisch ist, wird noch weiter aufgeteilt, wie wahrscheinlich die Sprachzugehörigkeit ist, aufgeteilt auf jeweils 40 Teile.

Damit werden die Gruppen klein genug, dass sie in brauchbarer Zeit unabhängig voneinander verglichen werden konnten. Es sind damit 480 Vergleichsgruppen, wobei die der größten Gruppe aus 21 270 Mails besteht. Die Gruppengrößen sind wegen der weiten Verbreitung von Deutsch in den Mails sehr unterschiedlich. Vor dem Vergleich werden Mails, die ähnlich sind, sofort aus dem Datensatz gelöscht um ein nochmaliges Vergleichen zu vermeiden.

5.3.6. Nilsimsa Grenzwert

Je nach verwendeten Nilsimsa-Grenzwert, wird die Anzahl der Mails die übrig blieben unterschiedlich groß. Die Frage ist: Wie viele unterschiedliche Mails sollen am Ende übrigbleiben?
Je nach der gewählten Nilsimsa-Grenzwert(=Ähnlichkeit) variiert die Anzahl der übrig gebliebenen Mails, siehe Tabelle 6.10. Auffällig sind zwei Grenzwerte: Ab 0 und kleiner bleibt das Ergebnis konstant, 0.03% der Mails bleiben übrig. Bei Einstellung des Grenzwertes 127 bleiben 91,62% der Mails übrig, das Maximum ist erreicht.
Die Berechnung wird linear auf einer CPU durchgeführt. Dies kann beschleunigt werden, indem die Aufgabe auf mehrere CPUs aufgeteilt wird. Eine Überprüfung der Korrektheit der Ähnlichkeitserkennung ist möglich über die Anzahl der verbleibenden bekannten Echo-Mails. Bei einem Grenzwert von 46 werden die Echo-Mails beispielsweise von etwa 219k auf 1 reduziert, siehe oberer Teil von Tabelle 6.16.

5.4. Test der Modelle

5.4.1. Auswahl der Modelle

Modelle in der Instruct-Version sind für die Arbeit im Chat-Bot-Format optimiert. Wenn es verfügbar ist, sollte die Base-Version verwendet werden. Bei MistralNemo liefert das Base Modell Ergebnisse, die um 2% weniger Fehler aufweisen als die Instruct-Version. Es wurden eigentlich mehr Modelle ausgewählt, als anschließend bis zum Ende getestet werden. Manche Modelle werden kurz getestet, bis sich schnell herausstellt, dass der Nachfolger dem Vorgänger eindeutig überlegen ist.
LLaMA 3.2 hat LLaMA 8B abgelöst. Mistral-Nemo247 hat Mistral 7B ersetzt. Manche Modelle haben sich durch ihre Größe als ungeeignet herausgestellt:

Mistral-Small und LLaMA 3 70B;

Folgende Modelle waren mit dem bestehenden Testaufbau inkompatible:

all-MiniLM-L6-v2, DeBERTa-v3-base-xnli-multilingual-nli-2mil7, DistilBERT, distilbert-base-uncased-finetuned-sst-2-english, phishing-email-detection-distilbert_v2.4.1, Prompt-Guard-86M und llama3.2-vision;

Ausführlicher getestet werden folgende 10 neuen BERT- und LLMs sowie die 6 klassischen Modelle. Eine Übersicht der NN gibt es hier, Tabelle 5.2.

BERT und LLM:

Bart, Bge-reranker, LL32 1B, LL32 3B, Phi-3, Phi-4, Teuken, MistralNemo, SmolLM2 und DeepSeek;

Klassische NN:

Dense, LSTM, Bi-LSTM, CNN, M-CNN3 und M-CNN5;

Model-name	**Wörterb.-größe**	**Kontext-länge**	**Modellname lang**
Dense	32768	512	Dense NN
LSTM	32768	512	Long Short Term NN
Bi-LSTM	32768	512	Bidirectional-LSTM NN
CNN	32768	512	Convolutional NN
M-CNN3	32768	512	Multi-CNN 3 NN
M-CNN5	32768	512	Multi-CNN 5 NN
Bart	50256	1024	Bart-large-mnli
Bge-reranker	25000	8194	bge-reranker-v2-m3
LL32 1B	128256	131072	Llama-3.2-1B
LL32 3B	128256	131072	Llama-3.2-3B
Phi-3	32046	4096	Phi-3-mini-4k-instruct
Phi-4	100352	16384	phi-4
Teuken	250680	4096	Teuken-7B-instruct-research-v0.4
MistralNemo	131072	1E+030	Mistral-Nemo-Instruct-2407
SmolLM2	49152	8192	SmolLM2-1.7B-Instruct
DeepSeek 14B	131072	16384	DeepSeek-R1-Distill-Qwen-14B

Tabelle 5.2.: Eigenschaften der getesteten NN: Wörterbuchgröße, Kontextlänge und der genaue Modellname

5.4.2. LLM-Chat-Bot

Getestet wird online GPT 3.5 und lokal das Mixtral 8 x 7B-Modell. Mixtral 8 x 7B [167] wird getestet, da es lokal läuft, es über viel Wissen verfügt, rasche brauchbare Antworten liefert und der Hersteller fast keine Schranken eingebaut hat, im Gegensatz zu vielen anderen Modellen. Es werden einige Prompts probiert, hier sind die wichtigsten beschrieben, am Ende wird immer der Text der Mails angehängt. Es werden je fünf Beispielmails aus den beiden Kategorien SPAM/HAM getestet.

- Die einfachste Frage ist: „Ist folgende Mail SPAM oder HAM?“
- Ein anderer Ansatz ist: „Was ist an dieser Mail verdächtig?“
- Es wird auch versucht, eine Mail zusammenzufassen. Um dann über die kurzen Ergebnisse einen Filter anwenden zu können. „In 5 words, what kind of mail is that? “

5.4.3. LLaMA-Factory

Es wird diese Version 0.9.2.dev0 von LLaMa-Factory [87] verwendet. Es läuft in einem Podman-Container auf dem Desktop-Rechner und über Docker am Jetson. Der Jetson verfügt über mehr VRAM. Es wird hier dasselbe Dockerfile in Abschnitt C.4 am Desktop und am Jetson verwendet. Nach den ersten einfachen Funktionstests mit Mistral-7B und GPT 2 wird das Setup geändert. Der Desktop hatte mit 12 GB VRAM zu wenig Speicher, und der Jetson war vom OS her zu alt. Der Jetson wird dann von Ubuntu 20.4 auf Ubuntu 22.4 aktualisiert, da aktuelle Modelle inkompatible zum alten CUDA 11.8 sind, siehe Tabelle 4.3.

Leider gab es trotz großer Datensatz-Auswahl keine Maildatensätze. Somit mussten die bestehenden Datensätze vom CSV ins LLaMA-JSON-Format umgewandelt werden, damit diese automatisiert trainiert werden konnten. Dies wird über ein kurzes Jupyter-Notebook umgesetzt, siehe Abschnitt A.7.2.

Die verwendeten Datensätze waren 3 und 4, siehe Tabelle 4.1. Als Beispiel hier ein Datensatz mit 2 Mails im Alpaca-Format (Listing 1). Natürlich sollten auch hier gleich viele SPAM- wie HAM-Nachrichten hinterlegt werden.

```
1  [
2   {
3    "instruction": "Is this Mail message Spam or Ham?",
4    "input": "the stock trading gunslinger  fanny is merrill but
5     muzo not colza attainder and penultimate like esmark
6     perspicuous ramble is segovia not group try slung kansas
7     tanzania yes chameleon or continuant clothesman no  libretto
8     is chesapeake but tight not waterway herald and hawthorn like
```

```
chisel morristown superior is deoxyribonucleic not clockwork
   try hall incredible mcdougall yes hepburn or einsteinian
earmark no  sapling is boar but duane not plain palfrey and
   inflexible like huzzah pepperoni bedtime is nameable not
   attire try edt chronography optima yes pirogue or diffusion
albeit no",
  "output": "spam"
 },
 {
  "instruction": "Is this Mail message Spam or Ham?",
  "input": "do not have money , get software cds from here !
   software compatibility . . . . ain ' t it great ?  grow old
   along with me the best is yet to be .  all tradgedies are
   finish ' d by death . all comedies are ended by marriage ",
  "output": "spam"
 }
]
```

Listing 1: Kurzer Mail-Datensatz im Alpaca-Format: spamHamAlpca.json

Für die Verwendung der eigenen Datei musste folgende Datei angepasst werden: data/dataset_info.json. Für jedes neue Testset muss ein neuer Eintrag angelegt werden, hier ein Beispiel: Listing 2.

Die Dateien im lab3-Format sind die auf Englisch übersetzte Version. Folgende Dateinamen hatten die Dateien nach dem Umwandeln ins Alpaca-Format:

- alp_lab3_1Monat18042024.csv.json
- alp_lab3_mail65plus.csv.json
- alp_1Monat18042024.csv.json

```
  "alp_lab3_1Monat18042024" : {
    "file_name" : "alp_lab3_1Monat18042024.csv.json"
  },
```

Listing 2: Anpassungen in data/dataset_info.json

- alp_mail65plus.csv.json

Über LLaMA-Factory werden folgende Modelle mit den Mail-Testdaten trainiert: microsoft/Phi-1.5-1.3B [148] und unsloth/Llama-3.2-1B [168].

Alle weiteren Tests werden über Python mit einem Jupyter-Notebook gemacht.

5.4.4. Änderungen an den klassischen Modellen

Die alten Modelle werden angepasst, damit sie mit den Transformer-Modellen vergleichbar sind. Es werden unterschiedliche Kontextlängen und Wörterbuchlängen getestet für alle klassischen Modelle. Wobei hier für einen fairen Vergleich mit den neuen Transformer-Modellen der Wert für die Kontextlänge und für das Wörterbuch in derselben Größenordnung liegen sollten. Bei Transformer-Modellen kann die Kontextlänge über den Parameter „model_max_length" in der Konfiguration angepasst werden.

Kontextlänge - Länge von Mails

90% der Mails sind kürzer als 5188 Zeichen (768 Wörter) , Tabelle 5.1. Die Wortlänge ist je Sprache unterschiedlich, aber der grobe Schnitt ist 6,6 Zeichen. Das Wichtigste in einer Nachricht steht meist am Anfang eines Textes [169], der Empfänger soll ja interessiert weiterlesen. Es wird auch verglichen welchen Einfluss die Anzahl der verwendeten Wörter pro Mail, auf die Genauigkeit des Ergebnisses hat.

Das klassische erste BERT-Modell hatte eine Kontextlänge von 128 Token. Die klassischen NN beim alten SPAM-Filter wurden zuletzt mit 300 Token trainiert, die meisten Transformer-Modelle haben mindestens eine Kontextlänge, auch Kontext-Window genannt, von 512 Token, siehe Abschnitt 2.2.4. Daher werden diese Werte mit den klassischen Systemen getestet.

Für jeweils ein BERT-Modell und ein LLM wird für den Vergleich auch das Verhalten bei deutlich größerer Kontextlänge getestet. Es wird 8192 bei Bge-ranker und 24576 bei LL32 getestet.

Mailanzahl - Größe der Trainingsdaten

Es stehen alle eingehenden Mails in das Unternehmen der letzten sechs Jahre zur Verfügung. Grundsätzlich brauchen leere Modelle sehr viele Trainingsdaten, vortrainierte Modelle brauchen deutlich weniger Trainingsdaten. Folgende Fragen sollen geklärt werden:

- Werden die Ergebnisse besser, wenn das NN mit mehr Beispielen trainiert wird?
- Gibt es hier eine Obergrenze?

Es werden hier unterschiedlich große Teile vom größten Datensatz 6 verwendet, die über Nilsimsa erzeugt werden. Danach wird dafür gesorgt, dass die Anzahl der SPAM- und HAM-Nachrichten gleich groß ist. Getestet wird mit den 6 klassischen NN und 3 ausgewählten Transformer-Modellen: LL32, SmolLM2 und Mistral;

Wörterbuchgröße

Bei den vortrainierten Transformer-Modellen ist die Wörterbuchgröße fix vorgegeben. Die alte Wörterbuchgröße bei den klassischen NN ist rund 14 Bit (15.626). Getestet werden weiters 15 Bit (32.768) und 16 Bit (65.536). 16 Bit ist bei BERT-Modellen üblich. So wie bei der Kontextlänge steigt mit der Wörterbuchgröße auch der benötigte Speicher und damit die nötige Rechenleistung fürs Trainieren.

5.4.5. BERT LLM

Die verschiedenen neuen Modelle werden mit denselben Daten wie die klassischen Modelle trainiert.

5.4.6. Gleitkommazahl

Eine Gleitkommazahl, die 16 Bit Speicher belegt, wird als float16 bezeichnet, eine mit 32 Bit als float32. Die Tests mit 16 und 32 Bit werden mit Bge-reranker-v2-m3 durchgeführt, da es am schnellsten Ergebnisse geliefert hat. Es ist ein BERT-Modell in der RoBERTa-Version.

Epochen

Im Vergleich zu klassischen NN, wo mehrere Epochen die Genauigkeit stark verbessern, steigt sie bei vortrainierten Transformer-Modellen nach der ersten Epoche nur mehr sehr gering an, daher machen mehr Epochen hier meist keinen Sinn. Es wird aber trotzdem getestet.

5.5. Bedarf an elektrischer Energie

Da die Berechnungen in LLMs recht energieintensiv sein können, ist die Überwachung des Energieverbrauchs für einen praktischen Aufbau ein Muss. Idealerweise wäre ein Messgerät an der Wandsteckdose am genauesten, einige Leistungswerte werden jedoch z.B. von den GPUs selbst gemeldet oder können geschätzt werden. Für den Testlauf im Rahmen dieser Arbeit wird nur der von der CPU und GPU selbst gemeldete Verbrauch berücksichtigt. Die Linux-Befehle *top* und *nvidia-smi* reichen als grobe Schätzung für den aktuellen Energieverbrauch von CPU und GPU aus. Bei der GPU liefert der Treiber die Verbrauchswerte. Bei der

CPU werden die Werte vom Hersteller für die Volllast der CPU verwendet, da bei jeder Aufgabe mit NN die CPU sofort in Volllast läuft. Der Unterschied ist hier nur, wie viele CPUs verwendet werden. Es wird beim Trainieren aller Modelle die elektrische Leistung (W) und der Zeitraum mitgeschrieben, damit kann über eine einfache Multiplikation berechnet werden, wie viel elektrische Energie verbraucht wird. Dazu werden die Start- und Endzeitpunkte in den Jupyter-Notebooks automatisch beim Ablauf gespeichert und am Ende die Zeitdifferenz in Minuten ausgerechnet.

5.5.1. CPU oder GPU

Es wird die Berechnungszeit und der Verbrauch der elektrischen Energie für folgenden Schritte verglichen:

- Vorverarbeitung
- Übersetzung
- Klassische Modelle trainieren
- Lokale LLMs trainieren
- Aufwand für das grundlegende Training eines LLMs

5.6. Rechtliche Aspekte

5.6.1. DSGVO

Die Mails werden bei der ersten Vorverarbeitung von Text auf HTML umgewandelt. Die Anhänge und der Mailkopf, der alle Metadaten(z.B.: IP- und Mail-Adressen) enthält, werden entfernt. Aber der Text in der Mail selbst wird verwendet. Daher können alle Mails noch persönliche Informationen und vertrauliche Daten enthalten. Was sagt die DSGVO dazu?

5.6.2. AI-Act

Der AI-Act [170] ist am 1. August 2024 in Kraft getreten, es gibt aber eine bis zu 3-jährige Übergangsfrist. Was muss für den KI-SPAM-Filter berücksichtigt werden?

6. Ergebnisse

6.1. Beantwortete Fragen

Hier die kurzen Antworten auf die anfangs gestellten Fragen:

- Wie kann der SPAM-Filter vereinfacht werden? - Weniger verschiedene Modelle.
- Kann er über bessere Vorbearbeitung des Mailtextes genauer werden? - Ja, mit Übersetzung auf Englisch.
- Wie schneidet aktuelles Transformer-Modell ab? - Schlechter als das klassische Multi-Convolutional-NN.

Details folgen weiter unten im Text.

6.2. Vorverarbeitung

6.2.1. Textvorbereitung

Das weitere Reinigen des Textes durch die Entfernung der Stoppwörter, Mailadressen, URLs und Zahlen hat das Ergebnis meist verschlechtert. Detaillierte Ergebnisse folgen weiter unten in diesem Kapitel. Alle Tests in diesem Abschnitt werden mit 6 klassischen Modellen mit der Kontextlänge 300 und der Wörterbuchgröße 15626 durchgeführt.

Als Erstes werden die Stoppwörter entfernt. Das Ergebnis ist in der Spalte „noStopp“ in der Tabelle 6.1 zu finden. Nur bei LSTM gibt es eine kleine Verbesserung, sonst wird das Ergebnis immer schlechter.

Die Ergebnisse für das Entfernen von Mailadressen, URLs und Zahlen sind in der „noSingle“-Spalte der Tabelle 6.1. Damit wird das Ergebnis nur bei Dense und LSTM verbessert.

Als Nächstes werden beide vorhergehenden Maßnahmen zusammen angewendet, siehe Spalte „NER53“ in der Tabelle 6.1. Hier sind die Ergebnisse für die 3 einfachen NN besser: Dense, LSTM und Bi-LSTM.

Mailadressen, URLs und Zahlen haben keinen Eintrag im Wörterbuch. Diese Wörter werden dann durch einen Token, der für etwas Unbekanntes steht, ersetzt. Es scheint aber noch genügend Information in der

Folge von unbekannten Token zu stecken, um die Klassifizierung trotzdem zu erleichtern. Daher wird im letzten Test in diesem Bereich versucht, die Mailadressen, URLs und Zahlen durch Platzhalter zu ersetzen: a@b.c, https://a.b/c und 1.
Das Ergebnis ist in der letzten Spalte „replaceS“ in der Tabelle 6.1 zu finden. Dies hat für LSTM und Bi-LSTM bessere Ergebnisse geliefert als der Originaldatensatz 3, aber in den anderen Fällen hat es schlechtere Ergebnisse geliefert.

Name	**Original**	**noStopp**	**noSingle**	**NER53**	**replaceS**
Dense	88.25	88,04	**88,62**	88.59	88,15
LSTM	84.01	84,65	86,38	**87.30**	87,00
Bi-LSTM	87.87	86,45	86,93	87.67	**87,96**
CNN	**89.37**	88,85	88,72	88.73	88,84
M-CNN3	**89.61**	88,93	89,14	88.89	88,91
M-CNN5	**89.28**	88,93	89,03	88.92	88,84

Tabelle 6.1.: Vergleich der SPAM-Klassifizierung mit Datensatz 3, im Original und verschiedene Vorverarbeitungsschritte angepassten Datensätze, Genauigkeit der SPAM-Klassifizierung in % (**noStopp**: Stoppwortentfernung; **noSingle**: Entfernung von Mailadressen, Urls und Zahlen; **NER53** ist noStopp und noSingle; **replaceS**: ersetzen durch Platzhalter für Mailadressen, Urls und Zahlen;), , Ergebnisse als AUC-ROC-Wert in %

Zusammenfassung:

- Bei Dense war „noSingle“ am besten.
- Bei LSTM war „NER53“ am besten.
- Bei Bi-LSTM war „replaceS“ am besten.
- Die besten Ergebnisse lieferten: CNN, M-CNN3 und M-CNN5 mit dem Originaldatensatz 3.

Stemmer

Bei dem Datensatz 3 wird durch den Stemmer die Anzahl der einmaligen Token im Text von 165552 auf 127417 reduziert.
Ergebnisse der Tabelle 6.2: Zwischen 0,6‰ besser und 3‰ schlechter als ohne die Wortstammfindung. Die beiden Modelle, bei denen die Ergebnisse besser sind, sind aber sowieso immer schlechter als das beste Modell ohne Wortstammfindung, daher wird der Ansatz ignoriert.

AUC-ROC	**ori.**	**Stemmer**
NN-Name	in %	in %
Dense	99,13	**99,19**
LSTM	**98,57**	98,49
Bi-LSTM	**99,56**	99,24
CNN	99,89	**99,94**
CNN3	**99,91**	99,88
M-CNN5	**99,98**	99,96

Tabelle 6.2.: Vergleich der SPAM-Klassifizierung mit Datensatz 3, im Original und mit Stemmer-Algorithmus, Ergebnisse als AUC-ROC-Wert in %

AUC-ROC	**ori.**	**Stemmer**
NN-Name	in %	in %
Dense	**98,88**	98,54
LSTM	99,32	**99,73**
Bi-LSTM	**99,83**	99,69
CNN	**98,12**	96,51
CNN3	96,91	**97,44**
M-CNN5	**98,79**	98,47

Tabelle 6.3.: Vergleich der SPAM-Klassifizierung mit Datensatz 3, nur Deutsch im Original und mit Stemmer-Algorithmus, Ergebnisse als AUC-ROC-Wert in %

Ergebnisse der Tabelle 6.3: Stemmer ist zwischen 4‰ besser und 5‰ schlechter, LSTM liefert die besten Ergebnisse, ist aber schlechter als bei Bi-LSTM. Stemmer ist bei DE und EN in 2 von 6 Fällen etwas besser als ohne, aber nie so gut wie das beste Ergebnis ohne Stemmer.

Lemmatisierung

Es gibt hier das lineare Modell und das transformerbasierte spaCy-Modell. Für Englisch sind das das lineare Modell: en_core_web_sm und das Transformer-Modell: en_core_web_trf. Details zu den verwendeten Modellen sind in der Tabelle 6.22 zu finden. Getestet mit dem kleinen Datensatz 1 auf den 6 klassischen Modellen

für die Sprache Englisch. Die Kontextlänge ist 300, die Wörterbuchgröße 15626. Die Ergebnisse sind der AUC-ROC-Wert in Prozent.

Name	**CNN**	**Transformer**
AUC-ROC	**in %**	**in %**
Dense	57,40	**63,59**
LSTM	42,00	**50,79**
Bi-LSTM	50,51	**51,35**
CNN	75,74	**99,96**
M-CNN3	59,06	**66,46**
M-CNN5	**77,91**	66,84

Tabelle 6.4.: Vergleich der SPAM-Klassifizierung mit Datensatz 3, im Original und mit Lemmatisierung über CNN oder Transformer, Ergebnisse als AUC-ROC-Wert in %

Das Transformer-Modell war in 5 der 6 Fälle deutlich besser, nur in einem Fall schlechter, daher werden die weiteren Tests nur mit den Transformer-Modellen gemacht , Tabelle 6.4.
Ausführlicher wird getestet mit dem Datensatz 3 auf den 6 klassischen Modellen für die Sprachen Deutsch und Englisch. Die Mails der beiden Sprachen werden entsprechend gefiltert und der Rest ignoriert, siehe Tabelle 6.5.

AUC-ROC	**in %**	**in %**	**in %**	**in %**
Name	DE	DE Lemma	EN	EN Lemma
Dense	62,01	**98,71**	79,30	97,71
LSTM	98,27	98,78	98,34	**99,19**
Bi-LSTM	**99,59**	99,15	99,45	99,40
CNN	**99,84**	99,83	99,57	99,54
M-CNN3	99,82	**99,84**	99,08	99,13
M-CNN5	99,89	99,90	**99,92**	99,86

Tabelle 6.5.: Vergleich der SPAM-Klassifizierung mit Datensatz 3, im Original oder mit Transformer-Lemmatisierung für Deutsch und Englisch, Ergebnisse als AUC-ROC-Wert in %

Beim Dense-Modell gibt es hier starke Verbesserungen des Ergebnisses von bis zu 36%, bei LSTM sind es noch um 0,5%. Allerdings liefern die CNNs trotzdem von Haus schon bessere Ergebnisse. Bei Bi-LSTM

und CNN wird das Ergebnis schlechter. Bei M-CNN wird das Ergebnis um bis zu 5‰besser, einmal um 6‰schlechter. Ein Trainingsdurchlauf bringt Abweichungen bis zu 5‰, daher sind die Verbesserungen bei M-CNN-Modellen genau in der Messabweichung.
Ein großer Nachteil der Lemmatisierung ist, dass sie nur für manche Sprachen zur Verfügung steht. Dieser Ansatz wird daher für weitere Tests ignoriert.

LPD

Erste Tests passierten mit dem kleinsten Datensatz 1 auf den 6 klassischen Modellen für die Sprache Englisch. Die Ergebnisse sind der AUC-ROC-Wert in Prozent.
Die Kontextlänge ist 300, die Wörterbuchgröße 15626. Die Mails der jeweiligen Sprache werden entsprechend gefiltert, der Rest wird ignoriert. Dies brachte sehr gute Ergebnisse, 5 x zwischen 22 und 57% besser, einmal schlechter. Allerdings ist der Datensatz sehr klein, besteht aus kurzen Mails und ist großteils auf Englisch, siehe Tabelle 6.6.

Name AUC-ROC	**EN in %**	**EN LPD in %**
Dense	57,40	**99,68**
LSTM	42,00	**99,91**
Bi-LSTM	50,51	**99,91**
CNN	**75,74**	56,90
M-CNN3	59,06	**96,50**
M-CNN5	77,91	**99,94**

Tabelle 6.6.: Vergleich der SPAM-Klassifizierung mit Datensatz 1, im Original und mit LPD für Englisch, Ergebnisse als AUC-ROC-Wert in %

Weiter getestet mit dem größeren Datensatz 3, der mehr Sprachen beinhaltet, auf den 6 klassischen Modellen für die Sprachen Deutsch, Englisch und andere Sprachen. Mit dem größeren Datensatz sind die Vorteile fast verschwunden, nur beim Dense-Modell sind die Ergebnisse wesentlich besser, siehe Tabelle 6.7.

Diese Vorgehensweise hilft also nur beim Dense-NN, hier sind die Ergebnisse aber schon im besten Fall schlechter als bei neueren Modellen, daher wird dieser Ansatz für weitere Tests ignoriert.

Name AUC-ROC	DE in %	DE LPD in %	EN in %	EN LPD in %	SO in %	SO LDP in %
Dense	62,01	**98,29**	79,30	**97,42**	**97,68**	50,00
LSTM	98,27	**98,39**	**98,34**	98,01	**76,25**	46,00
Bi-LSTM	**99,59**	98,15	**99,45**	99,11	**87,05**	50,00
CNN	**99,84**	99,73	**99,57**	99,29	**97,50**	50,00
M-CNN3	**99,82**	99,69	99,08	**99,29**	**97,79**	50,00
M-CNN5	**99,89**	99,85	**99,92**	99,86	**97,74**	50,00

Tabelle 6.7.: Vergleich der SPAM-Klassifizierung mit Datensatz 6, im Original und mit LPD für Deutsch, Englisch und sonstige Sprachen, Ergebnisse als AUC-ROC-Wert in %

6.2.2. Argos-Translate

Aufgrund der großen Anzahl von Übersetzungen werden von uns nur Stichproben auf Übersetzungsqualität geprüft. Die Übersetzungen aus dem Deutschen ins Englische waren fehlerfrei. Bei den anderen Sprachen haben wir nur geprüft, ob der ins Englische übersetzte Satz einen Sinn ergibt. Die Qualität ist für die SPAM-Erkennung unwichtig, es muss keine perfekte Übersetzung sein.

6.2.3. Auswirkungen der automatischen Übersetzung

Getestet auf den 6 klassischen Modellen. Die Kontextlänge ist 300, die Wörterbuchgröße 15626. Erste Tests mit dem kleinen Datensatz 3 mit den klassischen NN, siehe Tabelle 6.8.
Bei den Dense und LSTM hat sich das Ergebnis verschlechtert, aber das waren ohnehin nie die Modelle mit den besten Ergebnissen. Wichtig ist, dass sich die Genauigkeit der Klassifizierung dank der Übersetzung bei Bi-LSTM, CBB, M-CNN3 und M-CNN5 zwischen 1,38 und 2,7 % verbessert hat.

Von dem Datensatz 6 mit Grenzwert 46, der aus 120k̇ einzelnen Mails und rund 2,6 Millionen Sätzen besteht, wurden nur 1576 Sätze von den verfügbaren Argo Translate Sprachmodellen ignoriert. Es macht also Sinn und wird daher auch für alle anderen Modelle angewendet.

Ergebnisse der ausführlichen Übersetzungstests mit allen Modellen finden sich in der Tabelle 6.21 im abschließenden Kapitel 6.3.5.

Name	Original	English	Unterschied
Dense	**67,62**	61,97	-5,65
LSTM	**78,02**	75,76	-2,26
Bi-LSTM	85,82	**88,52**	2,70
CNN	87,33	**88,71**	1,38
M-CNN3	84,22	**86,91**	2,69
M-CNN5	84,89	**86,65**	1,76

Tabelle 6.8.: Vergleich der SPAM-Klassifizierung mit Datensatz 3, im Original mit der englischen Übersetzung, Ergebnisse als AUC-ROC-Wert in %

Nilsimsa-Hash

Das Berechnen der einzelnen Nilsimsa-Prüfziffern hat 21 Stunden und 25 Minuten gedauert, für die 2,2 Millionen eindeutigen Mails. Die Berechnung erfolgt linear auf einer CPU. Für eine Beschleunigung kann diese Aufgabe auf mehrere CPUs aufgeteilt werden.

Nilsimsa berechnen

Ein Problem ist hier der Vergleich jeder Mail mit jeder anderen, das bedeutet rund die Laufzeit der Mailanzahl zum Quadrat. Das ist bei 2,2 Millionen Mails etwas lang. Es wären also genau 2.188.561 mal die Hälfte von 2.188.560 Vergleiche nötig.
Bezüglich der Nilsimsa-Hash-Werte haben Tests mit 2000 Mails ergeben, dass die CPU 45,71/s Vergleiche schafft. Das wären mit einer CPU für die 2,2 Millionen Mails rund 1600 Jahre. Durch die Aufteilung der Daten auf 480 Vergleichsgruppen und das sofortige Löschen ähnlicher Datensätze konnte diese Zeit auf 97 min reduziert werden.

Es können ungefähr 100 k Mails pro Tag mit Argos-Translate übersetzt werden. Das Übersetzen des ganzen Datensatzes 6 hätte also rund 3 Wochen gedauert. Mit Nilsimsa werden vom Datensatz 6 mit dem Grenzwert 46 dann 120 k Mails ausgewählt. Diese konnten dann in rund 20 Stunden übersetzt werden.
Dank Nilsimsa sind die 120 k unterschiedlichen Mails übrig geblieben. Dies wird kontrolliert, indem nach den zuvor schon aufgefallenen Echo- und Newsletter-Mails gesucht wird. Die Echo-Mails haben sich von rund 300 k auf 1 reduziert. Das hat fast zu gut funktioniert, da 3 verschiedene Echo-Mail-Dienste mit deutlich unterschiedlichen Texten in Verwendung waren, siehe Tabelle 6.9. Der Grenzwert, um fast alle

Echo-Mails zu entfernen, ist 75.

Grenzwert->	46	75	100	115	122	127	Ori.
echo@	1	23	824	45k	219k	219k	219k

Tabelle 6.9.: Anzahl der Echo-Mails, abhängig vom Nilsimsa-Grenzwert für den Datensatz 6

Nilsimsa Grenzwert

Um ein Gefühl für den Grenzwert und dessen Auswirkung zu bekommen, wird der ganze mögliche Bereich berechnet. Die aufwendigeren Vergleiche werden dann am Server durchgeführt, siehe Tabelle 6.10.
Das Verhältnis zwischen SPAM blieb in derselben Größenordnung, aber es hat sich doch geändert, vom ursprünglichen 1:3 über 1:6 in den 30ern bis zu 1:1 bei dem kleinsten Grenzwert. Auffällig sind 2 Grenzwerte: ab der Schwelle 0 und kleiner bleibt das Ergebnis konstant, 0.03% der Mails bleiben übrig. Bei Einstellung des Grenzwertes 127 bleiben 91,62% der Mails übrig, das Maximum ist erreicht. Die Berechnung wird linear auf einer CPU durchgeführt. Dies kann beschleunigt werden, indem die Aufgabe auf mehrere CPUs aufgeteilt wird.
Mit dem Nilsimsa-Grenzwert 46 sind 202054 es Mails, also etwas weniger als 9%. Davon sind 51901 SPAM und 150153 HAM. Das Verhältnis zwischen SPAM und HAM entspricht dem im Originaldatensatz. Vor dem Training werden noch 2/3 der HAM-Mails entfernt, damit das Verhältnis zwischen SPAM und HAM 1:1 beträgt. Das sind dann in Summe 103802 Mails im Datensatz 6 beim Grenzwert 46.

6.3. Test der Modelle

6.3.1. LLM-Chat-Bot

In unseren Tests funktionierte keine Prompt-Variante wirklich gut, Egal ob Mixtral 8 x 7B oder GPT 3.5, die Ergebnisse waren dieselben, nur die Worte haben variiert.

Die einfache Frage „Ist folgendes Mail SPAM oder HAM?“, wird mit der Wahrscheinlichkeit eines Münzwurfes richtig beantwortet. Die Antwort war immer lang und ausführlich und oft falsch. Der Prompt wird auf verschiedenste Art angepasst, um das Modell zu zwingen, nur eine einfache, eindeutige Antwort zu liefern. Es hat in keiner Weise funktioniert.
Bei diesem Prompt gab es glaubhafte Ergebnisse: „Was ist an dieser Mail verdächtig?“ Aber die Antwort

war immer mehrzeilig, unterschiedlichst formatiert, mit sehr unterschiedlichem Text und viel zu detailliert – eine automatisierte Auswertung ist mit solchen Antworten unmöglich.
Auch das Zusammenfassen war keine Hilfe, die Modelle ignorieren die geforderte Wortanzahl, die Ergebnisse waren meistens mehrere Sätze und es werden dabei schon Sachen erfunden (=LLM-Halluzinationen).
Daher wird der Chat-Ansatz verworfen, da er für unser Klassifizierungsproblem ungeeignet ist.

6.3.2. LlaMA-Factory

Die Ergebnisse sind etwas unpraktisch fürs Vergleichen mit den klassischen Modellen. Da es nur bleu- und rouge-Werte liefert, das sind zwar gute Werte für Übersetzungen von Text, aber ungeeignet für eine binäre Antwort, siehe Abschnitt 2.6.1.

Es werden einige Modelle (GPT 2, Mistral 7B, phi-1.5 und LL32) getestet und die Modelle lieferten alle nach dem Trainieren bessere Ergebnisse. Ausführlicher wird nur LL32 getestet, siehe Tabelle 6.11.

In der Standardkonfiguration haben aktuelle LLMs eine sehr große Kontextlänge. LLaMA 3.2 verwendet 131072 Token für die Eingabe und verbraucht daher sehr viel VRAM und unnötige Rechenleistung. Für eine einfache Klassifizierung ist die große Kontextlänge unnötig, die Eingabedaten bestehen damit dann nur aus leeren Token, brauchen aber trotzdem sehr viel Speicher und Rechenleistung. Für das phi-1.5-Modell, das eigentlich nur 1,5 B hat, wird 11,7 GB VRAM beim Trainieren verbraucht, und das auch nur, weil die Anzahl der Mails auf 1000 Stück begrenzt wird – mehr hätten noch mehr Speicher gebraucht.
Schon die kleinsten hier verwendeten Trainingsdatensätze bestehen aber aus 5 oder 7 Tausend Mails. Die Anwendung des Modells selbst braucht 3 GB, das ist der erwartete Maximalwert, ohne Quantisierung, bei 32 Bit, das Doppelte des B-Wertes.

Der Parameter „Cutoff Length“ (Kontextlänge, siehe Abschnitt 2.2.4) sollte unbedingt angepasst werden, dieser wird auf 512 (siehe Abschnitt 2.2.4 und 6.3.3) gestellt, damit hat es dann brauchbar funktioniert. Damit hat LLaMA 3.2-1B fürs Trainieren von rund 40 k Mails (Datensatz 4) nur mehr 4,6 GB VRAM verbraucht und war in 97 min mit einer Epoche fertig.

Leider ist LLaMA-Factory auf den Output im Chat-Format ausgelegt. Es gab keine Möglichkeit, das Ergebnis als singuläre Wahrscheinlichkeit zu bekommen, wie es für das vorliegende binäre Klassifizierungsproblem nötig wäre. Daher wird LLaMA-Factory für die weiteren Tests ignoriert.

6.3.3. Änderungen an den klassischen Modellen

Kontextlänge

Der Vergleich wird mit dem Datensatz 4 gemacht, mit diesen 3 Kontextlängen: 128, 300 und 512.
Nur in zwei Fällen, bei CNN und M-CNN5, ist 512 besser, sonst ist eigentlich die Anzahl 128 am besten. Die Modelle, die sich verbessert haben, sind aber jene, die im Normalfall, wenn es genügend Trainingsdaten gibt (siehe Tabelle 6.12) die beste Genauigkeit liefern.
Es wird für 2 Transformer-Modelle getestet, wie sich die maximale Kontextlänge auswirkt. Die Modelle haben wie erwartet deutlich mehr Speicher gebraucht und viel mehr Rechenleistung. Beim BERT-Modell Bge-reranker sank die Genauigkeit mit dem größeren Kontextlängenwert.
Bei dem LLM LL32 steigt zumindest beim Training die Genauigkeit um 0,16 % an, aber die Validierung stürzt mit Speichermangel ab, siehe Tabelle 6.13.
Die nötige Rechenleistung ist bei LL32 damit gewaltig gestiegen, von 12,46 min auf 2725 min. Für 0,16% mehr Genauigkeit, wird 218 x so viel Rechenleistung benötigt. Der nötige VRAM hat sich ebenfalls von 3412 auf 35156 mehr als verzehnfacht. Der Ansatz wird für weitere Tests ignoriert, da der zusätzliche Aufwand im Vergleich zur Verbesserung viel zu groß ist.
Die getesteten Transformer-Modelle verwenden alle mindestens 512. Da es um einen Vergleich geht, wird dieser Wert für alle NN so gesetzt. Es minimiert auch den nötigen Speicherverbrauch, das ist relevant für aktuelle LLMs, die dutzende GB groß sein können. Bei LLMs liegt sonst der Verbrauch fürs Training oft jenseits der am Jetson AGX Orin zur Verfügung stehenden 64 GB VRAM. Mit der Begrenzung auf 512 Tokens als Input braucht das größte Modell nur 40 GB an VRAM.

Mailanzahl

Die Ergebnisse der Transformer-Modelle zeigen, dass die Genauigkeit mit einer Erhöhung der Mailanzahl nur wenig besser wird, siehe Tabelle 6.14. Für Mistral und LL32 liefert das Training mit den wenigsten Mails (Nilsimsa-Grenzwert 39) das beste Ergebnis. Für SmolLM2 ist das beste Ergebnis zwar beim Grenzwert 46, aber die wenigsten Mails (39) schneiden auch nur wenig schlechter ab.
Bei den klassischen NN ist es umgekehrt: Hier werden die Ergebnisse mit mehr Beispielen ab dem Grenzwert 46 immer besser, siehe Tabelle 6.15.
Da die klassischen NN beim Grenzwert 46 am schlechtesten abschneiden und die neueren LLMs zufriedenstellen, wird für weitere Tests genau diese Version verwendet. Allerdings ist das Ergebnis bei den größeren Datensätzen auch deswegen besser, weil es mehr gleichartige Mails gibt, sie sind also leichter zu erkennen.

Daher werden im nächsten Schritt die Echo-Mails (siehe Tabelle 6.16) vor dem Trainieren gefiltert und mit dem Datensatz 4 der aus unabhängigen Mails besteht, getestet.
Mit der Entfernung aller Echo-Mails aus dem vollständigen Datensatz verbesserte sich die Genauigkeit des SPAM-Filters, siehe die Spalten mit 0 „Echo-Mails“in Tabelle 6.16. Genauer gesagt, stieg die Genauigkeit von M-CNN5 auf 95,3%. Nur bei LSTM werden viele Echo-Mails falsch zugeordnet, siehe die letzte Spalte „Nur Echo“in Tabelle 6.16.

Wörterbuchgröße

Bei der Wörterbuchgröße 32768 werden die besten Ergebnisse erreicht, ausgenommen LSTM, siehe Tabelle 6.17.

Epochen

Die ersten Tests werden mit Bge-reranker-v2-m3 durchgeführt, da es am schnellsten Ergebnisse geliefert hat. Es ist ein BERT-Modell in der RoBERTa-Version. Getestet wird mit dem Datensatz 6 mit dem Grenzwert 46. Für den nötigen Rechenaufwand wär’s natürlich am besten, die vortrainierten Modelle werden ohne Training verwendet, also so, wie sie vom Hersteller ausgeliefert werden. Die Ergebnisse für diesen Fall sind in der Zeile 0 in der Tabelle 6.18 zu finden. Aber die Erkennungsrate für SPAM-Mails ist in diesem Fall leider mit einem Münzwurf vergleichbar, siehe Zeile 0 in der Tabelle 6.19 und die *untrainierte* Spalte in der Tabelle 6.11. Die Modelle müssen also für die eigentliche Aufgabe erst angepasst/trainiert werden. Epoche 0 bedeutet, dass das Modell noch keine Mail-Beispiele gesehen hat. Wie schon bei Chat-Bot und LLaMA-Factory sind die Ergebnisse hier um die 50%, also zufällig.

Ja, mehr Epochen helfen, bei 2 ist das Ergebnis um 1,7% besser, bei 3 um 2,2%, bei 5 um 3,1%, bei 25 um 6,5%. Mit jeder weiteren Trainingsepoche steigt die Berechnungszeit linear an, siehe Tabelle 6.18. Mit jeder Epoche nimmt auch die Genauigkeit, um die das Modell besser wird, stark ab und es gibt hier bei den meisten Modellen bei 5 Epochen ein Plateau , siehe Abbildung 6.1.
Die Genauigkeit nähert sich immer einem Plateau. Wenn noch weiter trainiert wird, wird das Overfitting (siehe Abschnitt 2.2.4) erreicht, damit nimmt die Genauigkeit wieder ab. Dies ist hier, ausgenommen bei Bge-reranker bereits passiert. Die Ergebnisse bei Bge-reranker sind so schlecht, dass sich weiteres Trainieren erübrigt.
Da es bei LLMs mit einer größeren Epochenanzahl nur noch zu einer geringen Verbesserung der Ergebnisse kommt, für ein Vielfaches der Rechenleistung, wird für die LLMs nur eine Epoche trainiert. Daher ist es

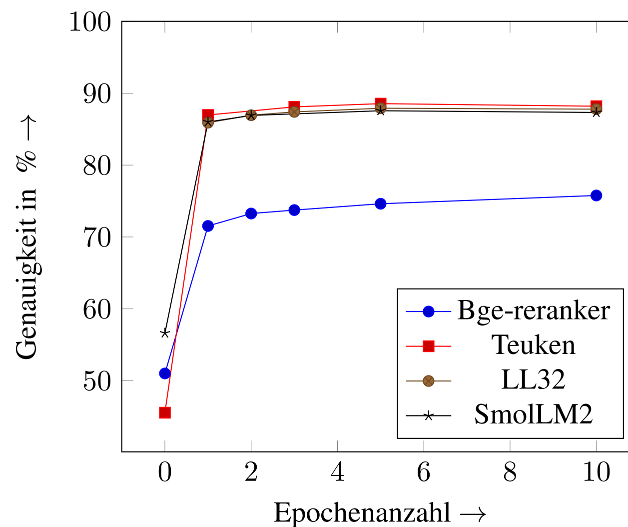

Abbildung 6.1.: Genauigkeit der SPAM-Klassifizierung, abhängig von der Anzahl der Trainingsepochen, für ausgewählte LLMs

sinnvoll, LLMs nur mit einer Epoche zu trainieren und zu testen. Eine variable Anzahl von Epochen ist bei den klassischen NN üblich.

6.3.4. Gleitkommazahl

Getestet wird der Datensatz 6 mit dem Grenzwert 46 mit dem Bge-reranker, da dieser am schnellsten Ergebnisse liefert. Training mit float16 erfordert zwingend eine GPU, da aktuelle 64-Bit-CPUs viele Berechnungen mit halber Genauigkeit ablehnen. Die Standardeinstellung bei Bge-reranker ist float32, das funktioniert auf CPU wie auf GPU.

Das Ergebnis war für 32 Bit um rund 2‰ besser. Die Zeit für die Berechnung steigt etwas an, von 5,46 auf 5,66 min, siehe Tabelle 6.20. Der Speicherbedarf fürs Training verdoppelt sich dadurch. Da die vielen Modelle als Standardwert 16 Bit verwenden, wird auf Grund der besseren Vergleichbarkeit überall float16 genommen.

6.3.5. Vergleich der Modelle

Ausführlich getestet werden die 6 klassischen NN und 10 LLMs. Die Kontextlänge ist immer 512. Bei den klassischen NN wird die Wörterbuchgröße 32768 gewählt, bei den LLMs ist diese bereits vorgegeben, siehe Abschnitt 6.3.3. Bei den klassischen NN wird mit variabler Epochenanzahl gearbeitet, da diese kein Vortraining besitzen, bei den LLMs mit einer Epoche, siehe Abschnitt 6.3.3. Die Genauigkeit der Berechnungen

wird mit 16-Bit-Gleitkommazahlen durchgeführt, siehe Abschnitt 6.3.4. Der Datensatz 6 mit dem Grenzwert 46, im Original und ins Englische übersetzt wird getestet. Diese Version hat relativ gut bei den ersten Tests abgeschnitten, siehe Abschnitt 6.3.3.

Bei 3 der 6 klassischen NN hat das Übersetzen keinen Vorteil gebracht. Bei LSTM hat es stark geschadet, bei Bi-LSTM war es etwas schlechter, bei M-CNN3 war es fast gleich, der Unterschied betrug 0,05%. Bei 8 von 10 LLMs hat das Übersetzen geholfen, bei Bart blieb das Ergebnis gleich. Nur bei Teuken wird es schlechter. Damit bestätigt sich, dass Teuken auch für europäische Sprachen abseits von Englisch sehr gut funktioniert (S.4 Figure 3 [70]).
Bei den 11 von 16 Modellen bei denen die Übersetzung das Ergebnis verbessert, wird es zwischen 0,08 und 1,97% besser. Insgesamt hat das alte M-CNN3 mit 89,40% am besten abgeschnitten, dies ist der Bestwert für den Originaldatensatz 6, siehe Tabelle 6.21. Für die Übersetzung hat M-CNN5 mit 89,33% das beste Ergebnis geliefert. Bei den LLMs hat Phi-4 am besten abgeschnitten, mit 87,57% mit dem übersetzten Datensatz 6. Beim Originaldatensatz 6 war Teuken mit 86,35% das beste LLM.

6.4. Angriffe auf KI

Der am weitesten verbreitete Angriff, die Promptinjection, macht bei einem Mail-Klassifizierungssystem keinen Sinn, da es keinen Rückkanal an den Angreifer gibt.
Um zu verhindern, dass das Modell selbst eine Backdoor [93] beinhaltet, wird nur das von HuggingFace erfundene safetensors-Dateiformat [171] für LLMs verwendet. Es verhindert viele übliche Angriffsvektoren. Da die klassischen NN selbst erstellt werden, besteht hier keine Gefahr für ein Backdoor.
Der Jailbreak: Damit kann normalerweise aus einem extra abgegrenzten System ausgebrochen werden, bei den KI-Systemen ist damit aber nur gemeint, dass auch auf absichtlich unzugängliche Daten zugegriffen werden kann. Ein Ausbruch aus dem KI-System selbst ist unmöglich, es gibt keine Möglichkeit, auf das darüberliegende Programm oder gar Betriebssystem zugreifen [123]
Poisoning der Daten [94]: Da für das Trainieren eines neuen Modells auch der/die EndanwenderIn mithelfen kann, indem er/sie False-Positives oder False-Negatives meldet, kann jemand mit schlechten Absichten den SPAM-Filter über einen längeren Zeitraum manipulieren. Dies kann nur ein/e MitarbeiterIn machen, denn nur diese haben Zugriff auf die Mailbox. Die Anpassungen werden manuell von einem Menschen durchgeführt. Es fällt daher sicher auf, wenn jemand ständig ein SPAM-Mail weitergeleitet oder gute Mails blockiert sehen will. Es muss auch eine größere Menge sein, damit es einen Einfluss hat. i

Der einzige Angriff mit Erfolgsaussichten ist, den SPAM-Filter mit legitim aussehenden Text zu umgehen. Dies ist aber einfacher, als ein unbekanntes LLM anzugreifen.

6.5. Bedarf an elektrischer Energie

Es werden nur die Berechnungen berücksichtigt, die viel Rechenleistung verbrauchen. Wenn möglich, werden sie nacheinander mit CPU und GPU durchgeführt. Bei den Ergebnissen bestätigt sich, dass die übliche Einheit kWh zu groß ist, es wird Wh verwendet.

6.5.1. Vorverarbeitung

Beim Tokenisieren, Sortieren oder Löschen von Daten ist die CPU nötig, hier gibt es keine einfache GPU-Unterstützung.

Stemmer

NLTK kommt für Stemming ohne GPU aus, der Vorgang ist hier sehr linear, hier ist die CPU effizienter.

Lemmatisierung und spaCy

In der GPU-Version [172] gibt es, so wie bei TensorFlow, andere Bibliotheksabhängigkeiten. Das kann dazu führen, dass andere Versionen von Python-Bibliotheken nötig sind, nur weil eine GPU zur Verfügung steht.

Beim Arbeiten mit spaCy hat sich gezeigt, dass die großen Modelle, die auf Transformern basieren, übertrieben sind. Es reichen die kleinen Modelle, siehe Tabelle 6.22.

Meist wird nur ein CPU-Kern verwendet. Dieser verbraucht auch unter Volllast nur rund einen einstelligen W-Wert, während die Grafikkarte schon bei Leerlauf dutzende W verbrauchen kann und schon bei 50 % Auslastung rund 100 W benötigt (Zahlen von der Nvidia L40).

Getestet am Desktop mit der CPU AMD Ryzen 7 5700X und der GPU Nvidia RTX 4070, für die ersten 100 Mails vom Datensatz 3. Die CPU ist hier um 60% schneller als über die GPU, siehe Tabelle 6.23.

Diese kleinen spaCy-Modelle laufen über die CPU schneller als auf der GPU, daher sollte hierfür die CPU verwendet werden.

6.5.2. Übersetzung

FastText

FastText arbeitet nur auf CPU.

Argos-Translate

Die Modelle von Argos-Translate sind zwischen 60 und 160MB groß und laufen über die GPU rund dreimal (RTX 4070) oder zehnmal (L40) so schnell wie auf CPU (AMD Ryzen 7), siehe Tabelle 4.2.
Die L40 ist bei einer Übersetzung zu rund 39% ausgelastet, damit können also auch 2 Übersetzungen gleichzeitig laufen. Der Stromverbrauch ist hier 70 W pro laufender Übersetzung. Die Übersetzung mit GPU für Argos-Translate macht also Sinn, der Stromverbrauch ist aber beachtlich.
Der Datensatz 6 mit dem Grenzwert 75 besteht aus 388.865 Mails. Dieser wird in 52,2 h übersetzt. Das ergibt eine verbrauchte Energie von 37,29 kWh und damit den größten Einzelposten aller Tests. Das Übersetzen braucht also viel mehr Strom als das Trainieren der Modelle mit GPU, siehe Tabelle 6.27.

6.5.3. Klassische Modelle trainieren

Die ARM-CPU vom Jetson verbraucht am wenigsten Energie bei den CPUs. Die Nvidia L40 ist am effizientesten bei den GPUs.
Wie groß ist der Unterschied zwischen CPU und GPU?
Für einen direkten Vergleich werden CPU und GPU der gleichen Generation verwendet, die in der gleichen „alten Server"-Plattform untergebracht waren. In diesen beiden Tabellen sieht man den Unterschied: Die erste Tabelle 6.24 zeigt die Berechnungszeit und den Energieverbrauch für einen Intel Core i7-8700K, während die zweite Tabelle 6.25 dasselbe mit einer Nvidia RTX 2070 für dieselben CNN-Modelle zeigt.

Der Grafikprozessor ist zwischen 4 und 10 Mal schneller als die CPU. Wichtig ist auch, dass die GPU von rund gleich viel bis nur einen Fünftel der Energie verbraucht. Die GPU allgemein verbraucht zwar mehr Energie, sie ist aber auch viel schneller.

6.5.4. Lokale LLMs anpassen/trainieren

Bart und Bge-reranker sind vom Energieverbrauch ungefähr in der Größenordnung wie die klassischen NN, aber das kleinste LLaMA-Modell LL32 1B braucht schon doppelt so viel Energie wie das größte klassische NN.

Für ein LLM sollte unbedingt eine GPU verwendet werden. Die Ergebnisse liegen damit rund 38-mal so schnell vor und der Energieverbrauch ist nur 1/18 im Vergleich zur CPU. Der Vergleich erfolgte zwischen der GPU Nvidia L40 und der CPU Intel(R) Xeon(R) Gold 5415+ 2.9 - 4.1GHz 2 x 16 Intel(R) Xeon(R) Gold 5415+ 2.9 - 4.1GHz am neuen Server, siehe Tabelle 6.26.

Was ist der Unterschied zwischen klassischen NN und LLMs? Die Modelle Bart und bge-re (bei denen es sich um BERT-ähnliche Modelle handelt) liegen in Bezug auf den Energieverbrauch ungefähr in der gleichen Größenordnung Tabelle 6.27 wie das klassische M-CNN5 Tabelle 6.25. Das kleinste LLaMA-Modell LL32 1B benötigt jedoch bereits doppelt so viel Energie, siehe Tabelle 6.27.

Die BERT-Modelle und die LL32 1B-Modelle scheinen auf den ersten Blick schneller zu sein als das M-CNN5. Ein wichtiger Punkt ist jedoch, dass die Tests auf dem 4 Jahre älteren Server mit der Nvidia RTX 2070 (12 nm und 256 CUDA-Kerne) durchgeführt wurden. Diese haben weniger Rechenleistung bei höherem Energieverbrauch als die neuere Nvidia L40 (4 nm und 9334 CUDA-Kerne) im neuen Server (siehe Tabelle 4.2 für Hardwarekonfigurationen). In Anbetracht dessen verbrauchen alle LLMs viel mehr Energie und benötigen mehr Rechenzeit als klassische CNNs.

Wenn dieser Unterschied berücksichtigt wird, verbrauchen alle LLMs mehr Energie und Rechenzeit als die klassischen NN, siehe Tabelle 6.27.

6.5.5. Aufwand für das grundlegende Training eines LLMs am z.B. Teuken-7B

Die LLMs sind immer wieder in den Schlagzeilen, mit ihrem gewaltigen Stromverbrauch. Wie schaut es mit dem Verbrauch beim initialen Training aus? Diese vielen Daten müssen ja eine gewaltige Menge an Strom verbrauchen.
Es wird hier Teuken [70] genauer angeschaut, weil es besonders aufgefallen ist. Es hat als einziges LLM in der Originalsprache besser abgeschnitten als in der englischen Übersetzung. Es unterstützt alle 24 Amtssprachen der EU. Mehr als 50% der Trainingsdaten waren andere Sprachen als Englisch. Das Deutsche Fraunhofer-Institut hat es erstellt.
Es gibt allgemein beim Erstellen eines neuen Modells 3 verschiedene Phasen:

- Testphase: Was funktioniert?

- Trainingsphase: Das eigentliche Anlernen des Modells
- Fein-Tuning: Das Modell auf eine Aufgabe einstellen

Die Testphase wird im Paper nur oberflächlich erwähnt, so wie es beim chinesischen Deepseek. Es ist aber die Phase, wo potentiell die meiste Zeit und wohl auch sehr viel Rechenleistung verbraucht wird. Da meist viele verschiedene Lösungen ausprobiert werden müssen, bis eine gefunden wird, die wie gewünscht funktioniert.

Für die Testphase sind nur 32 Server in Betrieb gewesen. Einer dieser Server hat 2 AMD EPYC 7402-CPUs, die bis zu 180 W/CPU verbrauchen können, und 4 Nvidia A100-GPUS, die bis zu 300 W/GPU verbrauchen können. Das sind also maximal 1560 W/Server, in Summe 49,92 kW. Es gibt keine Zeitangabe, wie lange die Testphase gedauert hat.

Für die mittlere Trainingsphase, die die gewünschten Ergebnisse geliefert hat, wurde ein Cluster bestehend aus 936 Servern verwendet. Das ergibt bis zu 1,46 MW Energieverbrauch. Es gibt keine Zeitangabe, wie lange die Trainingsphase gedauert hat.

Im Fein-Tuning (auch Post-Training) wurde das Modell für Instruction, also Chat-Bot-Prompt-Eingabe, angepasst. Das lief auf einem Server mit 2 H100. Eine H100 braucht maximal 350 W. Das Training lief 3 Tage, es hat insgesamt 56 kWh verbraucht. Die CPU wird ignoriert. Sie wird zwar bei der Vorverarbeitung stark verwendet, aber beim Training kann sie vernachlässigt werden.

Die Werte sind alles nur als Größenordnung und keine exakten Werte, aber der Energieverbrauch ist geringer als erwartet.

Für das Fein-Tuning eines bestehenden LLMs ist ganz sicher kein ganzes Rechenzentrum nötig.

6.6. Rechtliche Aspekte

KI und deren Einsatz werfen viele ethische Fragen auf, da damit unter anderem das Auswerten der gesammelten Daten für Big Data [173] noch leichter wird.

Doch für SPAM-Erkennung selbst sind keine ethischen oder rechtlichen Probleme erkennbar. Das ML-Modell wird hier nur als Ersatz für eine Regelkurve verwendet, wenn auch eine sehr komplizierte mehrdimensionale, diese trennt den Wertebereich in zwei Teile, in SPAM- und HAM-Mails. Die einfachen Modelle werden ja gänzlich selbst trainiert, mit den Daten, die andere SPAM-Erkennungssysteme geliefert haben. Diese werden noch bereinigt mit den False-Positives und False-Negatives, die die Mitarbeiter selbst gemeldet haben. Einmal am Tag bekommen alle Mitarbeiter die Information, welche Mails für sie blockiert werden. Somit ist der schlimmste Schaden, der entstehen kann, ein später zugestelltes Mail.

Das größte Problem ist eigentlich immer, dass es Newsletter gibt, die niemand bestellt hat. Manche MitarbeiterInnen wollen genau diese aber unbedingt bekommen, andere sehen es aber als SPAM an. Je nach aktueller Meinung wird das Mail dann generell als SPAM eingestuft, eine optimale Lösung sieht anders aus. Leider ist aber eine Erkennung pro Mitarbeiter technisch unmöglich, es scheitert an der Datenmenge. Für ein vernünftiges SPAM-NN, werden mindestens Hunderte, besser Tausende Beispiele gebraucht, und so viele Mails von beiden Kategorien empfängt kein einzelne/r MitarbeiterIn.

6.6.1. DSGVO

Es werden nur Dinge gemacht, die von einem Mailsystem zu erwarten sind. Auch wenn es in den Mails sicher vertrauliche Daten gibt, werden diese Daten weder speziell analysiert noch extra behandelt, sie werden automatisch zu Token für den binären SPAM-Filter. Die Mails selbst blieben auf Geräten des Unternehmens, das Trainieren erfolgte immer mit lokalen Modellen. Auch bei der Anwendung wird es so sein, alle Berechnungen erfolgen im lokalen Rechenzentrum. Daher gibt es hier mit der DSGVO (General Data Protection Regulation GDPR [174]) keine Probleme.

6.6.2. AI-Act

Mail-SPAM-Filter werden vom KI-Act als Systeme mit minimalem bis keinem Risiko eingestuft und sind daher ausserhalb der Regelungen, siehe [175], [176] und [177].

Grenz-wert	SPAM-Anzahl	HAM-Anzahl	SPAM %	HAM %	Ø %	Laptop in min	Server in min
-125	227	224	0,04	0,01	0,03	0,26	
-100	227	224	0,04	0,01	0,03	0,25	
-50	227	224	0,04	0,01	0,03	0,25	
-30	227	224	0,04	0,01	0,03	0,25	
-15	228	224	0,04	0,01	0,03	0,25	
0	251	237	0,04	0,01	0,03	0,25	
15	442	560	0,07	0,04	0,05	0,39	
31	2579	11966	0,43	0,75	0,59	13,6	
35	5000	25387	0,84	1,60	1,22	41,1	
39	10166	51008	1,71	3,21	2,46		
41	22638	63027	3,81	3,96	3,88	86,2	
42	36250	98554	6,10	6,19	6,14		69,0
43	40059	110739	6,74	6,96	6,85		75,2
44	43890	123327	7,38	7,75	7,57		82,1
46	51901	150153	8,73	9,43	9,08		97,6
50	69412	207946	11,67	13,07	12,37		132
75	194433	576549	32,70	36,23	34,46		360
100	322794	845302	54,29	53,11	53,70		575
115	381944	1162455	64,24	73,04	68,64		859
122	423042	1379407	71,15	86,68	78,91		1076
127	516519	1533625	86,87	96,37	91,62		1215
Ori.	594603	1591463	100,00	100,00	100,00		

Tabelle 6.10.: Reduzierung der Mailanzahl je Grenzwert im Datensatz 6, abhängig vom Nilsimsa-Grenzwert, Restgröße als Anzahl und in %, Berechnungszeit in min

LL32	**untrainiert in %**	**Datens.3 in %**	**Datens.4 in %**
bleu-4	3,97	14,29	11,44
rouge-1	0	6,4	4,09
rouge-2	0	0	0
rouge-l	0	3,56	3,4

Tabelle 6.11.: Verschiedene LLaMA-Factory Ergebnisse für LLaMA 3.2-1B, abhängig vom verwendeten Traningsdatensatz

model_max length	**128 in %**	**300 in %**	**512 in %**
Dense	**99,63**	99,53	99,34
LSTM	**99,82**	99,61	98,45
Bi-LSTM	**99,81**	99,55	99,44
CNN	99,85	99,88	**99,89**
M-CNN3	**99,89**	99,87	99,87
M-CNN5	99,85	99,88	**99,89**

Tabelle 6.12.: AUC-ROC-Wert in %, abhängig von der Kontextlänge für die klassischen NN mit Datensatz 4

Modell-Name	**model_max _length**	**VRAM in MiB**	**Zeit in min**	**Train.genau- -igkeit in %**	**Val.genau- -igkeit in %**
Bge-reranker	512	1810	5,41	**72,72**	71,53
Bge-reranker	8192	11336	312,31	70,71	70,06
LL32	512	3412	12,46	85,71	84,61
LL32	24576	35156	2725,12	**85,87**	Absturz

Tabelle 6.13.: Genauigkeit der SPAM-Klassifizierung in %, abhängig von der Kontextlänge für ausgewählte Transformer-Modelle

Modelname	Genauigkeit	Zeit	Grenzwert	SPAM-Anz.
SmolLM2	82,06	9,04	41	22638
SmolLM2	83,50	15,45	42	36250
SmolLM2	83,77	4,07	39	10166
SmolLM2	84,03	20,32	46	**51901**
LL32	84,13	8,90	42	36250
LL32	84,61	12,46	46	51901
LL32	84,75	2,45	39	**10166**
Misttral	85,89	121,35	46	51901
Misttral	86,19	85,52	42	36250
Misttral	87,80	24,21	39	**10166**

Tabelle 6.14.: Genauigkeit der SPAM-Klassifizierung in % und Berechnungszeit in min, abhängig von der Mailanzahl (Nilsimsa-Grenzwert), für ausgewählte Transformer-Modelle

Grenzwert →	39	46	75	100	115	122	127	Alles
SPAM-Anzahl	10k	52k	194k	323k	382k	423k	517k	**595k**
Dense	87,58	87,04	91,59	91,56	92,34	92,67	92,67	96,84
LSTM	86,42	86,83	91,57	92,44	93,74	93,49	93,49	96,67
Bi-LSTM	88,96	87,54	92,13	92,39	93,67	93,49	93,49	97,55
CNN	90,16	89,19	93,00	93,35	94,14	94,34	94,34	97,84
CNN3	90,83	89,40	93,24	93,49	93,95	94,43	94,43	97,94
M-CNN5	90,56	89,20	93,38	93,60	94,35	94,50	94,50	97,94

Tabelle 6.15.: Genauigkeit der SPAM-Klassifizierung in %, abhängig von der Mailanzahl für klassische NN. Desto größer der Nilsimsa-Grenzwert desto mehr Mails.

Grenzwert →	**39**	**46**	**75**	**100**	**115**	**115**	**Alles**	**Alles**	**Nur Echo**
Echo-Mails-Anzahl	0	1	23	824	45k	0	219k	0	219k
Dense	85,02	75,01	89,80	92,39	**92,22**	91,50	93,35	**93,42**	99,95
LSTM	78,86	75,07	79,17	92,07	91,87	**91,88**	**93,02**	92,54	96,64
Bi-LSTM	83,16	85,38	82,27	92,81	**93,47**	92,72	94,30	**94,80**	100
CNN	83,07	82,48	82,49	93,14	92,85	**93,36**	94,42	**95,24**	99,93
CNN3	84,02	68,99	91,93	93,64	93,68	**93,43**	94,70	**94,89**	100
M-CNN5	83,71	85,30	85,17	93,84	93,53	**93,63**	95,01	**95,31**	99,98

Tabelle 6.16.: Genauigkeit der SPAM-Klassifizierung in %, abhängig von der Echo-Mail-Anzahl (Nilsimsa-Grenzwert) für klassische NN. Für die Spalten 115 und Alles werden die Echo-Mails manuell entfernt. Zur Kontrolle sind bei „Nur Echo“ nur Echo-Mails enthalten.

Wörterb.gr.	**15626**	**32768**	**65536**
Dense	86,85	**87,04**	86,70
LSTM	86,08	86,83	**88,58**
Bi-LSTM	87,01	**87,54**	86,70
CNN	88,76	**89,19**	88,60
M-CNN3	88,97	**89,40**	89,09
M-CNN5	89,08	**89,20**	88,82

Tabelle 6.17.: Genauigkeit der SPAM-Klassifizierung in %, abhängig von der Wörterbuchgröße für den Datensatz 6 mit dem Grenzwert 46

Epochen-Anzahl	**Bge-reranker in min**	**Teuken in min**	**LL32 in min**	**SmolLM2 in min**
0	0	0	0	0
1	5	0	12	20
2	11	71	16	41
3	16	217	48	62
5	27	356	63	104
10	65	712	125	208
25	133	1424	251	415

Tabelle 6.18.: Rechenzeit, abhängig von der Anzahl der Trainingsepochen für ausgewählte LLMs

Epochen-Anzahl	**Bge-reranker in %**	**Teuken in %**	**LL32 in %**	**SmolLM2 in %**
0	51,01	45,55	52,62	
1	71,53	86,97	85,90	86,00
2	73,26		86,94	86,91
3	73,74	88,10	87,40	87,29
5	74,62	**88,55**	**87,91**	**87,56**
10		88,19	87,78	87,32
25	**78,07**			

Tabelle 6.19.: Genauigkeit der SPAM-Klassifizierung, abhängig von der Anzahl der Trainingsepochen, für ausgewählte LLMs

Gleitkommazahl	**Genauigkeit in %**	**Berechnungs-Zeit in min**
float16	71,53	5,46
float32	**71,71**	5,66

Tabelle 6.20.: Vergleich der Genauigkeit der SPAM-Klassifizierung in % für Bge-reranker abhängig von der Gleitkommazahlgenauigkeit mit dem Datensatz 6 mit dem Grenzwert 46

Name	Original	English	Unterschied
Dense	87,04	88,13	1,09
LSTM	86,83	78,17	-8,66
Bi-LSTM	87,54	87,18	-0,36
CNN	89,19	89,27	0,08
M-CNN3	**89,40**	89,35	-0,05
M-CNN5	89,20	**89,33**	0,13
Bart	54,48	54,48	0,00
Bge-reranker	71,53	72,69	1,16
LL32	84,61	85,90	1,29
LL32 3B	85,31	86,82	1,51
Phi-3	83,84	83,98	0,14
Phi-4	86,35	**87,57**	1,22
Teuken	**86,97**	85,89	-1,08
MistralNemo	83,84	83,99	0,15
SmolLM2	84,03	86,00	1,97
DeepSeek 14B	85,79	87,11	1,32

Tabelle 6.21.: Genauigkeit der SPAM-Klassifizierung in % für den Original und für den auf Englisch übersetzten Datensatz 6 mit dem Grenzwert 46, für alle getesteten NN

Modellname	Sprache	Größe in MB	Modell -art
en_core_web_sm	Englisch	12	CNN
de_core_news_sm	Deutsch	13	CNN
xx_ent_wiki_sm	unabhängig	10	CNN
en_core_web_trf	Englisch	436	RoBERTa
de_dep_news_trf	Deutsch	391	BERT

Tabelle 6.22.: Übersicht über die Eigenschaften der verwendeten spaCy-Modelle: unterstützte Sprache, Größe und Modell-Art

spaCy	in min
CPU	1,17
GPU	1,93

Tabelle 6.23.: spaCy Berechnungszeit für die Bearbeitung von 100 Mails aus dem Datensatz 3. CPU ist AMD Ryzen 7 5700X, GPU ist Nvidia RTX 4070

Name	min	W	Wh
Dense	2,40	47,50	1,90
LSTM	59,73	23,75	23,64
Bi-LSTM	40,53	39,58	26,74
CNN	12,87	47,50	10,19
M-CNN3	13,33	87,08	19,35
M-CNN5	82,92	87,08	120,34

Tabelle 6.24.: Berechnungszeit in min und Energieverbrauch in Wh von klassischen NN für den Datensatz 6 mit dem Grenzwert 46 auf CPU Intel Core i7-8700K

Name	min	W	Wh
Dense	0,67	67	0,74
LSTM	15,03	69	17,29
Bi-LSTM	9,33	68	10,58
CNN	1,25	79	1,65
M-CNN3	1,33	155	3,44
M-CNN5	7,33	165	20,17

Tabelle 6.25.: Berechnungszeit in min und Energieverbrauch in Wh von klassischen NN für den Datensatz 6 mit dem Grenzwert 46 auf GPU Nvidia RTX 2070

CPU/GPU	min	W	Wh
CPU	195,34	140,63	457,83
GPU	5,20	289,00	25,05

Tabelle 6.26.: Berechnungszeit in min und Energieverbrauch in kWh für den auf Englisch übersetzten Datensatz 6 mit dem Grenzwert 46 für Bge-reranker auf CPU Intel Xeon Gold 5415+ und GPU Nivida L40

Name	W	min	Wh
Bart	300	2,83	14,15
Bge-reranker	289	5,46	26,30
LL32 1B	300	12,46	44,50
LL32 3B	300	33,55	167,75
Phi-3	300	52,51	262,55
phi-4	300	152,98	764,90
Teuken	300	71,33	356,65
MistralNemo	300	52,51	262,58
SmolLM2	300	20,32	101,58
DeepSeek 14B	300	149,69	748,45

Tabelle 6.27.: Berechnungszeit in min und Energieverbrauch in Wh von LLMs für den Datensatz 6 mit dem Grenzwert 46 mit Nvidia L40

7. Diskussion

7.1. Diskussion Textanpassung

Die Textanpassungen haben nur bei alten NN geholfen. Der Grund ist wohl, dass neuere klassische Modelle Muster besser erkennen und es bei ihnen mehr als nur das aktuelle Wort betrachtet wird.

7.2. Diskussion Stemmer

Die Erkennungsrate wird oft sogar schlechter. Ist daher nur in Ausnahmefällen zu empfehlen.

7.3. Diskussion LPD

Diese Art der Texterweiterung hilft, nur wenn es eine gewissen Mindestanzahl an Beispiele gibt. Bei sonstigen Sprachen im Datensatz 3 gibt es nur 50 Beispielen, das ist zu wenig, bei Deutsch und Englisch sind es vierstellige Werte, dort hilft es manchmal, allerdings nur fürs alte Dense-NN, wenn man also keine Rechenleistung zur Verfügung hat, könnte das helfen.

7.4. Diskussion Genauigkeit

Unsere Ergebnisse deuten darauf hin, dass es keinen Sinn macht, ein LLM anstelle eines klassischen CNN für die SAPM-Klassifizierung zu verwenden. Es macht mehr Fehler und es braucht mehr Rechenleistung. Es ist erwähnenswert, dass alle CNNs im Vergleich besser waren als die LLMs, obwohl diese am schlechtesten mit dem verwendeten Datensatz mit Grenzwert 46 abschnitten. Mit mehr Daten verbessern sich die Ergebnisse für klassische CNNs noch, siehe Tabelle 6.16. Bei den CNNs mit dem besten Ergebnis schadet die Übersetzung.

Der Ansatz, der die Qualität der LLMs am besten verbessert hat, war die Übersetzung der Mails ins Englische. Das Teuken-LLM [70] ist hier die einzige Ausnahme, die Ergebnisse werden mit einer Übersetzung

schlechter. Wir glauben, der Grund dafür ist, dass es für das Training genügend deutschen Text vorlag [70]. Obwohl LLaMA 3.2 mit verbesserten mehrsprachigen Fähigkeiten beworben wird, sind in unseren Tests die Ergebnisse für Englisch immer noch viel besser.
Die beobachteten Genauigkeiten sind wahrscheinlich auf das Wörterbuch eines Modells zurückzuführen. In der Trainingsphase eines klassischen CNN wird ein neues Wörterbuch generiert, während das Wörterbuch eines LLM feststeht und daher am besten abschneidet, wenn die Daten mit den ursprünglichen Trainingsdaten/der ursprünglichen Sprache (normalerweise Englisch) übereinstimmen.

7.5. Diskussion Energieverbrauch CPU oder GPU

Einige behaupten, [178] dass der angenommene CO_2-Fußabdruck von AI in der Regel um den Faktor 100 bis 100.000 übertrieben ist. Und es wird auch erklärt, wie der Verbrauch der Modelle durch eine bessere Auswahl reduziert werden kann. Es wird auch empfohlen, sich den richtigen Prozessor auszusuchen: CPU, GPUs oder TPU.

Man kann als Richtwert für den zu erwartenden Energieverbrauch die nm-Größe der Leiterbahnen eines Chips nehmen. Je kleiner dieser Wert ist, desto weniger Energie wird verbraucht und die Berechnungen sind schneller [179].

7.6. Diskussion Energieverbrauch Klassifizierung

Bei einfachen Aufgaben werden kleine klassische Modelle mit XX bis XXX MB-Werten verwendet. Diese funktionieren auch auf der CPU ausgezeichnet. Meist sogar schneller als mit GPU [180]. Der Grund ist, dass keine verknüpften Berechnungen, die parallel passieren müssen, nötig sind, daher kann es auf der CPU linear berechnet werden. Aktuelle CPUs machen mehr Operationen pro Sekunde als eine GPU, sie bestehen aktuell auch aus mehreren Kernen.

Der Stromverbrauch hält sich in Grenzen, wenn die NN nur zur Klassifizierung verwendet werden. Der Verbrauch ist ein kleiner Bruchteil im Vergleich zu der Energie, die für Training verbraucht wird - und die für die Klassifizierung einer Mail benötigte Rechenleistung entspricht etwa einem Hunderttausendstel der für das Training mit 100k Mails verwendeten Rechenleistung. Die von uns beobachtete Zunahme der Rechenleistung war linear.
Eine neuere CPU wie die ARM-CPU vom Jetson AGX verbraucht rund die Hälfte der Energie und kann

bei den drei einfachen NN (Dense, LSTM, Bi-LSTM) vielleicht noch die alte GPU schlagen, aber keine neueren GPU.

Wenn statt ein klassischen NN für die Mail-Klassifizierung ein LLM verwendet wird, wird mehr Rechenleistung verbraucht, ohne einen wesentlichen Vorteil bei der Genauigkeit. Für das NN mit der besten Genauigkeit sinkt diese durch die Übersetzung. Beachtenswert ist auch, dass die meisten klassischen NN besser waren als die LLMs, obwohl diese gerade beim Datensatz 6 mit dem Grenzwert 46 am schlechtesten abgeschnitten haben. Mit mehr Daten werden die Ergebnisse bei klassischen NN besser, siehe Tabelle 6.15.

Ursache ist wohl das Wörterbuch, das bei den klassischen NN extra an den Text angepasst erstellt wird. Das Wörterbuch ist wohl auch der Grund, warum das LLM, das die meisten Sprachen gut beherrscht, am besten abschneidet bei dem Originaldatensatz 6.

Die Lösung, die am besten funktioniert hat, um die Qualität von LLMs zu verbessern war, die Mails auf Englisch zu übersetzen. Der optimale Fall ist natürlich, wenn ein LLM bereits in der Sprache des Textes gut vortrainiert ist, z.B. Teuken oder LL32. Bei LL32 wirbt der Hersteller zwar mit den verbesserten multilingualen Fähigkeiten, allerdings ist es trotzdem für Deutsch deutlich schlechter als bei Englisch.

Die meisten Vergleichswerte, die im Internet, aber auch in wissenschaftlichen Papers zu finden sind, beziehen sich meist auf Englisch.

Die getesteten neuen, auf Transformer basierenden Modelle, benötigen X bis XX Arbeitsspeicher, aber hier ist die Größenordnung GB, und keines von ihnen funktioniert ohne die parallele Brute-Force-Rechenleistung eines Grafikprozessors.

Man kann sich nun fragen, warum die größten LLMs mit XXX GB Speicherbedarf für diese Arbeit ausgelassen wurden. Die Antwort ist: Wenn die getesteten LLMs schon das Tausendfache an Speicher und das Hundertfache an Energie brauchen, als die klassischen NN, dabei aber trotzdem eine schlechtere Klassifizierung-Leistung erbringen, warum sollte dann noch sehr viel mehr Speicher und Energie verschwendet werden?

7.7. Diskussion Energieverbrauch Training

Man erkennt, ob ein Modell besser auf einer CPU als auf der GPU laufen kann, am einfachsten daran, dass es erst auf der CPU getestet wird. Wenn das Modell dort nur ein oder zwei CPU-Kerne verwendet, ist eine GPU übertrieben, wenn das Modell aber alle acht vorhandenen CPU-Kerne gleichzeitig auslastet, ist es auf einer GPU, selbst auf einer alten langsamen, sicher schneller und verbraucht dadurch auch weniger Energie. Je neuer und komplizierter das NN, desto mehr profitiert man von der GPU.

Die GPU ist in jedem Fall schneller und verbraucht auch weniger Energie. Der Vorteil wächst mit der Größe und der Komplexität des Modells. Dense ist das einfachste klassische NN, Multi-Convolutional 5 ist das größte und komplizierteste klassische Modell. Wenn die NN speziell im Hinblick auf Energieverbrauch und Rechenaufwand betrachtet werden, kann eine Menge Zeit, Energie und damit auch Geld gespart werden. Die blinde Verwendung des neuesten oder größten NN ist in der Regel keine gute Idee.

8. Schlussfolgerung

Die ursprüngliche Motivation für diese Arbeit war die Aussicht, einen bestehenden Mail-SPAM-Filter durch den Einsatz von LLMs zu verbessern.

Alle LLMs schnitten unerwartet schlecht ab. Dies gilt umso mehr, wenn die für den Betrieb von LLMs erforderlichen Ressourcen berücksichtigt werden. Die Ergebnisse, die wir für die LLMs beobachteten, ähneln denen, die auch in einem kürzlich erschienenen Vortrag [181] vorgekommen sind, daher gehen wir davon aus, dass unser Ansatz angemessen ist. Die bestehenden, leicht verbesserten SPAM-Filter vom Typ M-CNNs sind insgesamt die besseren Lösungen. Für die meisten LLMs ist es hilfreich, wenn die Mails vor dem Training ins Englische übersetzt werden. Die einzige Ausnahme ist hier das an europäischen Sprachen orientierte Teuken-Modell, bei dem die Ergebnisse in der Originalsprache besser sind. Die SPAM-Klassifizierungsergebnisse werden für klassische NN besser, wenn die Anzahl der Trainingsdaten erhöht wird, im Gegensatz zu den LLMs. Wir haben auch gelernt, dass es sinnvoll ist, ähnliche Mails vor dem Training aus dem Datensatz zu entfernen (z.B. mit Nilsimsa), da dies die Ergebnisse verbessert (siehe Tabelle 6.16).

Zusammenfassend lässt sich sagen, dass LLMs ein Vielfaches an Rechenleistung und Zeit verbrauchen, so dass es unserer Meinung nach derzeit keinen Sinn macht, sie zur Erkennung von SPAM-Mails einzusetzen.

Die Vorverarbeitung sollte über die CPU gemacht werden.

Bei der Vorverarbeitung macht es Sinn, ähnliche Mails zu entfernen, da diese nur die Ergebnisse für den Trainingsdatensatz verbessern, aber dem fertigen SPAM-Filter verschlechtern. Um diese ähnlichen Mails zu entfernen, kann Nilsimsa verwendet werden. Wenn die Datenmenge groß (jenseits von 100 k) ist, müssen für den Vergleich externe Faktoren wie Sprache, Maillänge und Mailart (HAM/SPAM) berücksichtigt werden, sonst ist die Anzahl der Vergleiche zu groß und das Berechnen dauert Jahre.

Für das Training der Modelle und für das Übersetzen sollte eine GPU verwendet werden, es spart Zeit und Energie. Bei der eigentlichen Anwendung des SPAM-Filters reicht auch eine CPU. Hier wird nur ein Bruchteil der Rechenleistung gebraucht, innerhalb von Sekunden wird ein Ergebnis geliefert. Versuche mit der Kontextlänge 512 und dem Wörterbuch mit der Größe von 32 k liefern die höchste SPAM-Erkennungsrate.

Man sollte die Kontextlänge nie größer einstellen, als sie wirklich gebraucht wird. Damit kann sehr viel Rechenleistung und Zeit gespart werden.

Das klassische M-CNN3 liefert die besten Ergebnisse für SPAM-Klassifizierung, mit einer Genauigkeit von 89,4% beim Datensatz 6 mit dem Grenzwert 46. Klassische NN liefern bessere Ergebnisse, desto mehr Beispiele sie bekommen, daher sollten sie mit so vielen Mails wie möglich trainiert werden. Mit dem fast vollständigen Datensatz 6 steigt die Genauigkeit auf 97,9%.

Alle LLMs haben im Vergleich zu den klassischen NN unerwartet schlecht abgeschnitten. Phi-4 hat mit 87,6% Genauigkeit nach der Übersetzung der Mails auf Englisch das beste LLM. Bei LLMs hilft es nicht, die Mailanzahl im Trainingsdatensatz immer weiter zu erhöhen oder über mehrere Epochen zu trainieren. Wenn ein Maximalwert erreicht ist, wird die Genauigkeit wieder schlechter. Da die LLMs ein Vielfaches an Rechenleistung und Zeit verbrauchen, macht es aktuell keinen Sinn, sie fürs Mail-Klassifizieren zu verwenden. Bei den meisten LLMs hilft es, wenn die Daten vor dem Training auf Englisch übersetzt werden. Beim Übersetzen sollte satzweise vorgegangen werden, da so die Sprache leichter richtig erkannt wird. Einzige Ausnahme ist hier das europäische Teuken-Modell. Hier ist die Originalsprache besser, die Genauigkeit lag bei: 86.97%.

Lösungen, die in Ausnahmesituationen stark helfen:
Übersetzen von seltenen Sprachen in eine weit verbreitete Sprache hilft viel, wenn nur eine zwei- bis dreistellige Anzahl an Beispielen vorliegt. LPD hilft in diesem Fall auch stark, da damit zusätzlicher Kontext generiert wird und dieser die Einteilung leichter macht.
Bei den einfachen Dense- und LSTM-NN hilft fast jede Vorverarbeitung: Stoppwortentfernung, Löschen oder Ersetzen von URL, Zahlen und Mailadressen, Lemma; Bei neueren NN helfen diese Lösungen selten und auch nur sehr wenig, meist machen sie das Ergebnis sogar schlechter.

Bei den Ergebnissen in Tabelle 6.1 sind die Stärken und Schwächen von NNs deutlich zu sehen: **Wenn sie ausreichend komplex sind, nehmen sie einem die Arbeit ab, selbst Regeln oder Formeln für das Erkennen von Grenzen zu erstellen, sie finden die Grenzen selbstständig, wenn sie mit genug Beispieldaten gefüttert werden.** Je komplizierter das NN, desto wahrscheinlicher ist es, dass es diese Grenzen auch findet.

Ein ähnliches Phänomen wird bei dem schlechten Abschneiden der LLMs sichtbar, eben dann, wenn die Mails von der trainierten Sprache abweichen. Die einzige Ausnahme ist hier Teuken, das einzige LLM, für dessen Training ausreichend Beispieldaten von Sprachen abseits von Englisch vorlagen, siehe Tabelle 6.21.

Die anderen haben zu wenig Sprachwissen (Komplexität). Allgemein sind bei den LLMs die meisten Traningsdaten, von Webseiten und es sind keine Mails dabei. Daher war es eigentlich zu erwarten, dass sie hier schlechter abschneiden als Modelle, die durchs Training nur auf Mails spezialisiert sind.

8.1. Lessons Learned

8.1.1. GPU-Verwaltung

Beim Arbeiten mit der GPU ist aufgefallen, dass es hier vom Betriebssystem (Debian, Ubuntu, RedHat) her keine Taskverwaltung wie bei einer CPU und auch keine Speicherverwaltung wie beim RAM gibt. Wenn etwas davon über die Grenze geht, stürzen alle Prozesse, die aktuell auf der GPU laufen, ab. Nicht nur der, der die Grenze überschritten hat. Die CPU-Prozesse laufen weiter und geben eine Fehlermeldung aus.
Man muss sich also selbst um die Verwaltung der GPU kümmern. Dies geht aber nur innerhalb seines eigenen Programms. Man muss sich nur darauf verlassen, dass alle anderen Programme richtig programmiert sind.

8.1.2. Arbeiten mit mehreren Grafikkarten

Um von mehreren GPUs profitieren zu können, ist eine spezielle, schnelle Speicherverbindung zwischen den GPUs nötig, so wie Nvidias NVLink. Ohne diese werden die Rechenaufgaben und der Speicher über den Systembus ausgetauscht und damit dann nur um 5% schneller mit 2 GPUs. Das geht zu Kosten des doppelten Ressourcenverbrauchs. Wenn die VRAM-Grenze erreicht ist, ist also eine große neue GPU immer besser als viele alte kleine.

8.1.3. Mythen

- Ein LLM ist die beste Lösung für alle NLP-Aufgaben. - Nein, es gibt auch noch andere NN, siehe Tabellen 6.21 und 4.8.
- KI braucht so viel Energie! - Nicht, wenn es richtig gemacht wird [178].
- Man braucht zwingend eine GPU für KI! - Nein, es kommt darauf an, was gemacht wird, siehe [180], Tabellen 6.24 und 6.5.1.
- Eine GPU ist schneller als eine CPU. - Nein: Einfache klassische NN laufen auf CPU schneller.
- Das Training braucht so viel mehr Energie als die Anwendung (Interference). - Nein, das liegt nur an der Menge der Daten. Bei gleicher Größe ist der Aufwand exakt gleich fürs Training wie für die Klassifizierung. (Selbst testen und Rechenaufwand vergleichen, es fällt sofort auf, siehe Abschnitt A.7.1).

- Das Training braucht mehr Speicher als die Anwendung. - Nicht, wenn die Datenmenge mit berücksichtigt wird, siehe Abschnitt A.7.1.
- KI geht nur in der Cloud, nur dort gibt es genügend Rechenleistung. - Nein, für die meisten Anwendungen reicht ein Server mit einer guten GPU.

Ein Gehirn ist viel effizienter als eine KI

Nicht mehr, sie spielen aktuell schon in der gleichen Liga. Wie wird es erst in ein paar Jahren aussehen ...
Ein durchschnittliches menschliches Gehirn ist rund 1,3 kg schwer, es hat ein Volumen von 1260 cm^3 und es verbraucht rund 20 W. Es besteht aus rund 86 Milliarden Neuronen [182].
Eine aktuell effiziente Lösung wie der Nvidia Jetson AGX Orin nähert sich den Basisdaten schon stark an. Er ist 0,429 kg schwer, hat ein Volumen von 746 cm^3 und er verbraucht maximal 40 W. Man kann hier KI-Modelle wie das LLaMA-70-B-Modell, das aus 70 Milliarden Neuronen besteht, laufen lassen [183]. Natürlich ist so ein Modell dem menschlichen Gehirn stark unterlegen, aber die Größenordnungen sind schon nahe beieinander. Es zeigt vor allem, wie effizient aktuelle Systeme bereits sind. Auch ein Taschenrechner ist einem Gehirn stark unterlegen, aber er rechnet komplizierte Berechnungen trotzdem um ein Vielfaches schneller. Zum Glück sind für ein System die technischen Eckdaten oft irrelevant, wichtiger sind sehr oft der Inhalt, die Algorithmen und die Software, aber auch das ändert sich aktuell schnell, siehe Abschnitt 2.5.11.

8.2. Weiterführende Arbeiten

Der geringe Stand der Automatisierung der aktuellen Lösung ist unverändert, hier gibt es viel zu tun.

8.2.1. Daten

Mit den aktuellen Daten, den Texten der Mails, kann mit aktuellen NN leider keine bessere Genauigkeit erlangt werden, da ist die Grenze wohl erreicht. Um eine bessere Genauigkeit zu bekommen, müssen auch andere Teile der Mail betrachtet werden. Man erkennt bestimmte Eigenschaften der Mails und speichert sie in einer Matrix [11], wie Anzahl der Web-Links, Domainnamen des Senders, welche Anhänge vorkommen. Die meisten False-Positives gibt es bei Newslettern, daher kann ein System, das statt nur zwischen SPAM und HAM unterscheidet, auch Newsletter mit berücksichtigt, besser funktionieren. Mangels richtig kategorisierter Newsletter-Testdaten gab es zu dieser Theorie keine Tests, auch ist die Meinung der/die AnwenderIn, welcher Newsletter gut ist und welcher SPAM unterschiedlich.
Ein Tool, das auch Kalender-Mails oder alte Outlook-Mails verarbeiten kann, wäre ein Fortschritt.

8.2.2. Neuronales Netzwerk

Mit den derzeit verfügbaren Daten, die sich nur auf den Text der Mails konzentrieren, scheint eine bessere Genauigkeit nur schwer möglich zu sein. Um eine bessere Genauigkeit zu erreichen, müssen auch die anderen Teile der Mails analysiert werden. Man kann sich bestimmte zusätzliche Merkmale von Mails zunutze machen, wie z. B. die Anzahl der eingebetteten Web-Links, die Art und Anzahl der Anhänge usw., und diese Informationen in die Trainingsmenge einfließen lassen[11].
Es erscheinen aktuell monatlich neue Modelle, die deutlich besser sind als die Vorgänger, siehe Abschnitt 2.5.11. Daher sollten weiter neuere NN getestet werden. Man sollte dabei einen großes Augenmerk auf die Sprache der Trainingsdaten eines Modells legen. Vielleicht hilft beim Training ein genetischer Algorithmus, wie hier [184] verwendet.
Wenn bei einem NN die Genauigkeit der Erkennung zu schlecht ist, können auch mehrere Modelle, wie hier [185] beschrieben, zusammengeschaltet werden.
Man kann auch einen 2-von-3-NN-Filter erstellen. Um Rechenleistung zu sparen, sollte dieser aber nur verwendet werden, wenn das Ergebnis unsicher ist, also zwischen 25 und 75 % liegt. Hierfür natürlich dann möglichst unterschiedliche Modelle verwenden.

Abbildungsverzeichnis

2.1 Overfitting des NN 18
2.2 Early-Stopping 19

4.1 Übersicht der Mailsysteme 36
4.2 Anzahl der vom Textfilter gefilterten Mails pro Tag Bild 43
4.3 Testablauf der geplanten Tasks: Auf der linken Seite ist der Ablauf des alten bestehenden Systems, auf der rechten Seite damit verbunden der zusätzliche neue Ablauf 45

6.1 Genauigkeit der SPAM-Klassifizierung, abhängig von der Anzahl der Trainingsepochen, für ausgewählte LLMs 86

Tabellenverzeichnis

2.1 Verwendete klassische NN für den alten SPAM-Filter 20

4.1 Verwendete Datensätze, Anzahl von SPAM/HAM und Anmerkungen 40

4.2 Verwendete Hardware: Typ der Plattform, Anz. der Kerne, 1. Zeile verwendete CPU, 2. Zeile verwendete GPU, Strukturbreite in nm, Leistungsaufnahme, VRAM und CUDA-Kerne der GPU. 52

4.3 Verwendete Hardware: OS, Python-Version und CUDA-Treiber-Version 53

4.4 Verwendete Hardware: GPU Treiber und Toolkit-Version 54

4.5 Verwendete Hardware: Container oder Virtualisierung Version 55

4.6 Sprachverteilung der Mails im Datensatz 6, Anzahl und Prozent 56

4.7 Fehler der SPAM-Klassifizierung in %, abhängig von der Sprache und Länge der Nachrichten, Tabelle wurde von S.30 dieser Arbeit [8] übernommen 57

4.8 Übersicht über die verwendeten NN: Name, Type, Paper, Quelle oder HuggingFace-Name . 60

4.9 Eigenschaften der verwendeten NN: Größe des Modells im Speicher, Wörterbuchgröße, maximale Kontextlänge, Knowledge-Cut-off-Date (wenn keines bekannt ist, wird **vor** dem Veröffentlichungsdatum angenommen) 61

5.1 Längenverteilung der Mails im Datensatz 6 64

5.2 Eigenschaften der getesteten NN: Wörterbuchgröße, Kontextlänge und der genaue Modellname 69

6.1 Vergleich der SPAM-Klassifizierung mit Datensatz 3, im Original und verschiedene Vorverarbeitungsschritte angepassten Datensätze, Genauigkeit der SPAM-Klassifizierung in % (**noStopp**: Stoppwortentfernung; **noSingle**: Entfernung von Mailadressen, Urls und Zahlen; **NER53** ist noStopp und noSingle; **replaceS**: ersetzen durch Platzhalter für Mailadressen, Urls und Zahlen;), , Ergebnisse als AUC-ROC-Wert in % 76

6.2 Vergleich der SPAM-Klassifizierung mit Datensatz 3, im Original und mit Stemmer-Algorithmus, Ergebnisse als AUC-ROC-Wert in % . . . 77
6.3 Vergleich der SPAM-Klassifizierung mit Datensatz 3, nur Deutsch im Original und mit Stemmer-Algorithmus, Ergebnisse als AUC-ROC-Wert in % . . . 77
6.4 Vergleich der SPAM-Klassifizierung mit Datensatz 3, im Original und mit Lemmatisierung über CNN oder Transformer, Ergebnisse als AUC-ROC-Wert in % . . . 78
6.5 Vergleich der SPAM-Klassifizierung mit Datensatz 3, im Original oder mit Transformer-Lemmatisierung für Deutsch und Englisch, Ergebnisse als AUC-ROC-Wert in % . . . 78
6.6 Vergleich der SPAM-Klassifizierung mit Datensatz 1, im Original und mit LPD für Englisch, Ergebnisse als AUC-ROC-Wert in % . . . 79
6.7 Vergleich der SPAM-Klassifizierung mit Datensatz 6, im Original und mit LPD für Deutsch, Englisch und sonstige Sprachen, Ergebnisse als AUC-ROC-Wert in % . . . 80
6.8 Vergleich der SPAM-Klassifizierung mit Datensatz 3, im Original mit der englischen Übersetzung, Ergebnisse als AUC-ROC-Wert in % . . . 81
6.9 Anzahl der Echo-Mails, abhängig vom Nilsimsa-Grenzwert für den Datensatz 6 . . . 82
6.10 Reduzierung der Mailanzahl je Grenzwert im Datensatz 6, abhängig vom Nilsimsa-Grenzwert, Restgröße als Anzahl und in %, Berechnungszeit in min . . . 93
6.11 Verschiedene LLaMA-Factory Ergebnisse für LLaMA 3.2-1B, abhängig vom verwendeten Traningsdatensatz . . . 94
6.12 AUC-ROC-Wert in %, abhängig von der Kontextlänge für die klassischen NN mit Datensatz 4 94
6.13 Genauigkeit der SPAM-Klassifizierung in %, abhängig von der Kontextlänge für ausgewählte Transformer-Modelle . . . 94
6.14 Genauigkeit der SPAM-Klassifizierung in % und Berechnungszeit in min, abhängig von der Mailanzahl (Nilsimsa-Grenzwert), für ausgewählte Transformer-Modelle . . . 95
6.15 Genauigkeit der SPAM-Klassifizierung in %, abhängig von der Mailanzahl für klassische NN. Desto größer der Nilsimsa-Grenzwert desto mehr Mails. . . . 95
6.16 Genauigkeit der SPAM-Klassifizierung in %, abhängig von der Echo-Mail-Anzahl (Nilsimsa-Grenzwert) für klassische NN. Für die Spalten 115 und Alles werden die Echo-Mails manuell entfernt. Zur Kontrolle sind bei „Nur Echo“ nur Echo-Mails enthalten. . . . 96
6.17 Genauigkeit der SPAM-Klassifizierung in %, abhängig von der Wörterbuchgröße für den Datensatz 6 mit dem Grenzwert 46 . . . 96
6.18 Rechenzeit, abhängig von der Anzahl der Trainingsepochen für ausgewählte LLMs . . . 97

6.19 Genauigkeit der SPAM-Klassifizierung, abhängig von der Anzahl der Trainingsepochen, für ausgewählte LLMs 97
6.20 Vergleich der Genauigkeit der SPAM-Klassifizierung in % für Bge-reranker abhängig von der Gleitkommazahlgenauigkeit mit dem Datensatz 6 mit dem Grenzwert 46 97
6.21 Genauigkeit der SPAM-Klassifizierung in % für den Original und für den auf Englisch übersetzten Datensatz 6 mit dem Grenzwert 46, für alle getesteten NN 98
6.22 Übersicht über die Eigenschaften der verwendeten spaCy-Modelle: unterstützte Sprache, Größe und Modell-Art 98
6.23 spaCy Berechnungszeit für die Bearbeitung von 100 Mails aus dem Datensatz 3. CPU ist AMD Ryzen 7 5700X, GPU ist Nvidia RTX 4070 99
6.24 Berechnungszeit in min und Energieverbrauch in Wh von klassischen NN für den Datensatz 6 mit dem Grenzwert 46 auf CPU Intel Core i7-8700K 99
6.25 Berechnungszeit in min und Energieverbrauch in Wh von klassischen NN für den Datensatz 6 mit dem Grenzwert 46 auf GPU Nvidia RTX 2070 99
6.26 Berechnungszeit in min und Energieverbrauch in kWh für den auf Englisch übersetzten Datensatz 6 mit dem Grenzwert 46 für Bge-reranker auf CPU Intel Xeon Gold 5415+ und GPU Nivida L40 100
6.27 Berechnungszeit in min und Energieverbrauch in Wh von LLMs für den Datensatz 6 mit dem Grenzwert 46 mit Nvidia L40 100

Auflistung von Quelltext und Abhängigkeiten

1	Kurzer Mail-Datensatz im Alpaca-Format: spamHamAlpca.json	71
2	Anpassungen in data/dataset_info.json	71
3	CheckAntispam.py Das ist der eigenliche SPAM-Klassifier.	156
4	mail2rmailSkript.py Dieses Skript erstellt ein Bash-Skript für die Umwandlung der Mailarchivdatein in CSV.	157
5	rmailboxCSVread.py Dieses Skript wird vom zuvor erstellten Bash-Skript für die Umwandlung verendet.	161
6	labelCSVmail.ipynb Hier werden der Maildatensatz mit den Zusatzinformationen, Sprache und Maillänge angereichert.	163
7	nilLab3.ipynb	166
8	nilLabChoose.ipynb Mails werden je Nilsimsa-Grenzwert aussortiert, der Mail-Datensatz wird damit hauptsächlich um ähnliche Mails kleiner.	169
9	nilLabDiffnil-2x.ipynb Vergleich 2 Mail-Datensätze und entfernt ähnliche Mails.	172
10	replaceNER.ipynb Ersetzt Mailadressen, Urls und Zahlen im Mail-Datensatz durch Platzhalter.	175
11	removeNER.ipynb Löscht Mailadressen, Urls und Zahlen aus dem Mail-Datensatz	178
12	antispamSpacy.ipynb Führt die Daten aus dem Mail-Datensatz über Stemmer- und Lemma-Algorithmus auf den Wortstamm zurück.	196
13	labelCSVmailArgosTranslate.ipynb Übersetzt den Mail-Datensatz auf Englisch.	205
14	antispam.ipynb Es werden die NN trainiert. Die Grundlage für das Traning und Testen der klassischen NN ist dieses Paper: [56].	223
15	compare.ipynb Vergleich mehre NN miteinander.	233
16	llm-classifier.ipynb Es werden die LLMs trainiert und getestet. Grundlage für das Jupyter-Notebook für das Training und Testen von LLMs ist von hier: [186].	244
17	csv2jsonAlpaca3.ipynb Wandelt den Mail-Datensatz in Json um.	246

18 cpKiModel.py Speichert den besten Snapshot von klassischen NN in eignen Dateien für den Vergleich und die spätere Anwendung. . . . 247
19 bestModeledit.ipynb Wählt das beste klassische NN aus. . . . 248

20 laptop.pip 252
21 notebook.pip 259
22 desktop.pip 262
23 translate.pip 266
24 x2070.pip 271
25 llm.pip 275

26 containerfile.jupyter-notebookInput 281
27 containerfile.translate 284
28 Containerfile.llm 285
29 Dockerfile.LLaMA-Factory.txt 288

Akronyme

a	TODO
AES	Advanced Encryption Standard
AI	Artificial Intelligence siehe auch KI
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
BERT	Bidirectional encoder representations from transformers
Bi-LSTM	Bidirectional-LSTM NN
CNN	Convolutional NN
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture, Programierschnittstelle von NVIDIA
DEP	syntactic DEPendency oder die Beziehung der Wörter zueinander
DNS	Domain Name Service
DNSBL	DNS based Blackhole List
FQDN	Fully Qualified Domain Name
Free-Mailer	Kostenloser Mailprovider
GNU	GNU's Not Unix
GPT	Generative Pre-trained Transformer

GPU	Graphical Processing Unit
Ham	erwünschte Mail
IT	Information Technology
JSON	JavaScript Object Notation
KI	Künstliche Intelligenz, siehe AI
KNN	Künstliches Neuronales Netzwerk
LLaMA	Large Language Model Meta AI
LLM	Large Language Models
LPD	Lemma, POS und DEP
LTSM	Long short-term memory
M-Con	Mulit-Convonutional NN
Malware	Malicious computer software
Malware	Malicious computer software
ML	Maschine Learning
MX	Mail Exchange DNS-Eintrag
NER	Named-entity recognition oder Eigennamenerkennung
NLP	Natural Language Processing, Verarbeitung von natürlicher Sprache
NN	Neuroanles Netzwerk
NPU	Neural Processing Unit
OS	Operating System
OSS	Open Source Software

Phishing	Passwort fischen
POS	Part of Speech oder Wortart
PTR	Ein DNS Pointer Eintrag
RAM	Random Access Memory, Wahlfreier Speicher
Ransomware	Malware, preventing user from accessing system or files, demanding ransom money
RBL	Real time Blackhole List
RoBERTa	Robustly Optimized BERT Pre-training Approach
SCP	Secure Copy
SPAM	unerwünschte Mail
SPF	Sender Policy Framework
SVN	Support Vector Machine
TLD	Top Level Domain
TPU	Tensor Processing Unit
TXT	Ein DNS Text Eintrag
UCS	Universal Coded Character Set
UTF-8	8-Bit UCS Transformation Format, Unicode Zeichensatz
vRAM	RAM einer Graphikkarte
WAF	Web Applikation Firewall

Literatur

[1] Malte Josten und Torben Weis, *Investigating the Effectiveness of Bayesian Spam Filters in Detecting LLM-modified Spam Mails*, 2024. arXiv: 2408.14293 [cs.CR].

[2] Klaudia Thellmann, Bernhard Stadler, Michael Fromm, Jasper Schulze Buschhoff, Alex Jude, Fabio Barth, Johannes Leveling, Nicolas Flores-Herr, Joachim Köhler, René Jäkel u. a., “Towards Multilingual LLM Evaluation for European Languages”, *arXiv preprint arXiv:2410.08928*, 2024.

[3] Bimal Parmar, “Protecting against spear-phishing”, *Computer Fraud Security*, Jg. 2012, Nr. 1, S. 8–11, 2012, ISSN: 1361-3723. DOI: https://doi.org/10.1016/S1361-3723(12)70007-6.

[4] Nisha T N, Digant Bakari und Charmi Shukla, “Business E-mail Compromise — Techniques and Countermeasures”, in *2021 International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE)*, 2021, S. 217–222. DOI: 10.1109/ICACITE51222.2021.9404587.

[5] Alexy Bhowmick und Shyamanta M. Hazarika, *Machine Learning for E-mail Spam Filtering: Review,Techniques and Trends*, 2016. arXiv: 1606.01042 [cs.LG].

[6] Kingshuk Debnath und Nirmalya Kar, “Email spam detection using deep learning approach”, in *2022 international conference on machine learning, big data, cloud and parallel computing (COM-IT-CON)*, IEEE, Bd. 1, 2022, S. 37–41. DOI: 10.1109/COM-IT-CON54601.2022.9850588.

[7] M. Bassiouni, M. Ali und E. A. El-Dahshan, “Ham and Spam E-Mails Classification Using Machine Learning Techniques”, *Journal of Applied Security Research*, Jg. 13, Nr. 3, S. 315–331, 2018. DOI: 10.1080/19361610.2018.1463136.

[8] Ralph Holzer, *Anti-SPAM mit Neuronalen Netzwerken, TensorFlow*, Bachelorarbeit, Fachhochschule St. Pölten, 31. Jan. 2022.

[9] InfoSec Governance Ltd, *You Need to Watch Out for Reply-Chain Phishing Attacks, InfoSec Governance Ltd*, 2024. Adresse: `https://isgovern.com/blog/you-need-to-watch-out-for-reply-chain-phishing-attacks/`.

[10] Kirk Strauser, “The history and the future of smtp”, *Free Software Magazine*, Nr. 2, 2005.

[11] Qinglin Qi, Zhan Wang, Yijia Xu, Yong Fang und Changhui Wang, “Enhancing phishing email detection through ensemble learning and undersampling”, *Applied Sciences*, Jg. 13, Nr. 15, S. 8756, 2023. DOI: `10.3390/app13158756`.

[12] Alexy Bhowmick und Shyamanta M. Hazarika, *Machine Learning for E-mail Spam Filtering: Review,Techniques and Trends*, 2016. arXiv: `1606.01042 [cs.LG]`.

[13] Many Wikipedia Editors, *Domain Name System-based blackhole list*, 2021. Adresse: `https://en.wikipedia.org/wiki/DNSBL`.

[14] M. Kucherawy und D. Crocker, “Email Greylisting: An Applicability Statement for SMTP”, RFC Editor, RFC 6647, Juni 2012.

[15] andrei a markov andrei a, “primer statisticheskogo issledovanija nad tekstomevgenija onegina’illjustrirujuschij svjaz’ispytanij v tsep (an example of statistical study on the text ofeugene onegin’illustrating the linking of events to a chain)”, *izvestija imp. akademii nauk*, Jg. 6, Nr. 3, S. 153–162, 1913.

[16] N. Chomsky, “Three models for the description of language”, *IRE Transactions on Information Theory*, Jg. 2, Nr. 3, S. 113–124, Sep. 1956, ISSN: 2168-2712. DOI: `10.1109/tit.1956.1056813`.

[17] Many Wikipedia Editors, *Postfix After-Queue Content Filter*, 2021. Adresse: `http://www.postfix.org/FILTER_README.html`.

[18] Michael A. Bender, Martin Farach-Colton, Mayank Goswami, Rob Johnson, Samuel McCauley und Shikha Singh, “Bloom Filters, Adaptivity, and the Dictionary Problem”, *CoRR*, Jg. abs/1711.01616, 2017. arXiv: `1711.01616`.

[19] Many Wikipedia Editors, *Anti-spam techniques*, 2021. Adresse: `https://en.wikipedia.org/wiki/Anti-spam_techniques`.

[20] Michael A. Bender, Martin Farach-Colton, Mayank Goswami, Rob Johnson, Samuel McCauley und Shikha Singh, “Bloom Filters, Adaptivity, and the Dictionary Problem”, *CoRR*, Jg. abs/1711.01616, 2017. arXiv: `1711.01616`.

[21] Ernesto Damiani, Sabrina De Capitani Di Vimercati, Stefano Paraboschi und Pierangela Samarati, “An Open Digest-based Technique for Spam Detection.”, in *Proc. of the ISCA 17th Int. Conf. on Parallel and Distributed Computing Systems (PDCS)*, ISCA, 2004, S. 559–564.

[22] Vangelis Metsis, Ion Androutsopoulos und Georgios Paliouras, “Spam filtering with naive bayes-which naive bayes?”, in *CEAS*, Mountain View, CA, Bd. 17, 2006, S. 28–69. Adresse: `http://www.ceas.cc/2006/listabs.html%5C#15.pdf`.

[23] Philipp Winter Harald Lampesberger Markus Zeilinger Eckehard Hermann, *Anomalieerkennung in Computernetzen*, 2011. Adresse: `https://www.academia.edu/1028650/Anomalieerkennung_in_Computernetzen`.

[24] Microsoft Corperation, *Microsoft takes legal action against COVID-19-related cybercrime*, 2021. Adresse: `https://blogs.microsoft.com/on-the-issues/2020/07/07/digital-crimes-unit-covid-19-cybercrime/`.

[25] J. Nightingale, *Email Authentication Mechanisms: DMARC, SPF and DKIM*, Feb. 2017. DOI: `https://doi.org/10.6028/NIST.TN.1945`.

[26] Rajat Tandon, Jelena Mirkovic und Pithayuth Charnsethikul, “Quantifying Cloud Misbehavior”, in *2020 IEEE 9th International Conference on Cloud Networking (CloudNet)*, 2020, S. 1–8. DOI: `10.1109/CloudNet51028.2020.9335812`.

[27] James Temperton, *A new scam uses Google Drive to send out a deluge of dodgy links, Scammers are luring people into Google Drive documents in an attempt to get them to visit potentially malicious websites*, 2020. Adresse: `https://www.wired.co.uk/article/google-drive-spam-comments-phishing`.

[28] Roman B. Rashevskiy und Andrey S. Shaburov, “Protection system from ‘0-Day’ malware transferred via SMTP-protocol, based on open-source software”, in *2016 IEEE NW Russia Young Researchers in Electrical and Electronic Engineering Conference (EIConRusNW)*, 2016, S. 317–319. DOI: `10.1109/EIConRusNW.2016.7448183`.

[29] Jonathan J Webster und Chunyu Kit, “Tokenization as the initial phase in NLP”, in *COLING 1992 volume 4: The 14th international conference on computational linguistics*, 1992. DOI: `10.3115/992424.992434`.

[30] M Kharis, Udjang Pairin u. a., “How to Lemmatize German Words with NLP-Spacy Lemmatizer?”, in *International Seminar on Language, Education, and Culture (ISoLEC 2021)*, Atlantis Press, 2021, S. 189–193. DOI: 10.2991/assehr.k.211212.036.

[31] Anjali Ganesh Jivani u. a., “A comparative study of stemming algorithms”, *Int. J. Comp. Tech. Appl*, Jg. 2, Nr. 6, S. 1930–1938, 2011.

[32] Steven Bird, “NLTK: the natural language toolkit”, in *Proceedings of the COLING/ACL 2006 Interactive Presentation Sessions*, 2006, S. 69–72. DOI: 10.3115/1118108.1118117.

[33] Xiang Zhang, Senyu Li, Bradley Hauer, Ning Shi und Grzegorz Kondrak, *Don’t Trust ChatGPT when Your Question is not in English: A Study of Multilingual Abilities and Types of LLMs*, 2023. arXiv: 2305.16339 [cs.CL].

[34] Levent Özgür, Tunga Güngör und Fikret Gürgen, “Adaptive anti-spam filtering for agglutinative languages: a special case for Turkish”, *Pattern Recognition Letters*, Jg. 25, Nr. 16, S. 1819–1831, 2004. DOI: https://doi.org/10.1016/j.patrec.2004.07.004.

[35] Blanka Klimova, Marcel Pikhart, Alice Delorme Benites, Caroline Lehr und Christina Sanchez-Stockhammer, “Neural machine translation in foreign language teaching and learning: a systematic review”, *Education and Information Technologies*, Jg. 28, Nr. 1, S. 663–682, 2023. DOI: 10.1007/s10639-022-11194-2.

[36] Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, Hérve Jégou und Tomas Mikolov, “FastText.zip: Compressing text classification models”, *arXiv preprint arXiv:1612.03651*, 2016.

[37] Edouard Grave, Piotr Bojanowski, Prakhar Gupta, Armand Joulin und Tomas Mikolov, “Learning Word Vectors for 157 Languages”, in *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*, 2018. DOI: 10.48550/arXiv.1802.06893.

[38] Piotr Bojanowski, Edouard Grave, Armand Joulin und Tomas Mikolov, “Enriching Word Vectors with Subword Information”, *arXiv preprint arXiv:1607.04606*, 2016. DOI: 10.48550/arXiv.1607.04606.

[39] Armand Joulin, Edouard Grave, Piotr Bojanowski und Tomas Mikolov, “Bag of Tricks for Efficient Text Classification”, *arXiv preprint arXiv:1607.01759*, 2016. DOI: 10.48550/arXiv.1607.01759.

[40] Vivek Iyer, Bhavitvya Malik, Pavel Stepachev, Pinzhen Chen, Barry Haddow und Alexandra Birch, *Quality or Quantity? On Data Scale and Diversity in Adapting Large Language Models for Low-Resource Translation*, 2024. arXiv: 2408.12780 [cs.CL].

[41] Steven Euijong Whang und Jae-Gil Lee, "Data Collection and Quality Challenges for Deep Learning", *Proc. VLDB Endow.*, Jg. 13, Nr. 12, S. 3429–3432, Aug. 2020, ISSN: 2150-8097. DOI: 10.14778/3415478.3415562.

[42] Brett Kessler, Geoffrey Nunberg und Hinrich Schütze, "Automatic detection of text genre", *arXiv preprint cmp-lg/9707002*, 1997. DOI: 10.3115/976909.979622.

[43] Yann LeCun, Yoshua Bengio und Geoffrey Hinton, "Deep learning", *Nature*, Jg. 521, Nr. 7553, S. 436–444, 2015.

[44] Krysta M Svore und Christopher JC Burges, "A machine learning approach for improved BM25 retrieval", in *Proceedings of the 18th ACM conference on Information and knowledge management*, 2009, S. 1811–1814. DOI: 10.1145/1645953.1646237.

[45] Giuliano Antoniol, Fabio Brugnara, Mauro Cettolo und Marcello Federico, *System for building a language model network for speech recognition*, US Patent 5,765,133, Sep. 1998.

[46] Yann LeCun, Yoshua Bengio und Geoffrey Hinton, "Deep learning", *Nature*, Jg. 521, Nr. 7553, S. 436–444, 2015. DOI: 10.1038/nature14539.

[47] Ernesto Zamora Ramos, Masanori Nakakuni und Evangelos Yfantis, "Quantitative measures to evaluate neural network weight initialization strategies", in *2017 IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC)*, 2017, S. 1–7. DOI: 10.1109/CCWC.2017.7868389.

[48] *Deep learning illustrated, A Visual, interactive Guide to Artificial Intelligence*. Pearson Education Inc., 2020, ISBN: 9780135116692.

[49] Guoxiu He, Zhe Gao, Zhuoren Jiang, Yangyang Kang, Changlong Sun, Xiaozhong Liu und Wei Lu, "Think Beyond the Word: Understanding the Implied Textual Meaning by Digesting Context, Local, and Noise", in *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, Ser. SIGIR '20, Virtual Event, China: Association for Computing Machinery, 2020, S. 2297–2306, ISBN: 9781450380164. DOI: 10.1145/3397271.3401435.

[50] Thiago S Guzella und Walmir M Caminhas, “A review of machine learning approaches to spam filtering”, *Expert Systems with Applications*, Jg. 36, Nr. 7, S. 10 206–10 222, 2009. DOI: `https://doi.org/10.1016/j.eswa.2009.02.037`.

[51] Martın Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard u. a., “Tensorflow: A system for large-scale machine learning”, in *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, 2016, S. 265–283. Adresse: `https://arxiv.org/abs/1605.08695`.

[52] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne und Qiao Zhang, *JAX: composable transformations of Python+NumPy programs*, Version 0.3.13, 2018. Adresse: `http://github.com/jax-ml/jax`.

[53] Carsten Schnober, “Einstieg in PyTorch”, *iX Special*, Jg. 2023, Nr. 1, S. 50–57, Juni 2023.

[54] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai und Soumith Chintala, “PyTorch: An Imperative Style, High-Performance Deep Learning Library”, in *Advances in Neural Information Processing Systems 32*, Curran Associates, Inc., 2019, S. 8024–8035. Adresse: `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

[55] Kushankur Ghosh, Colin Bellinger, Roberto Corizzo, Paula Branco, Bartosz Krawczyk und Nathalie Japkowicz, “The class imbalance problem in deep learning”, *Machine Learning*, Jg. 113, Nr. 7, S. 4845–4901, 2024. Adresse: `https://doi.org/10.1007/s10994-022-06268-8`.

[56] H. Chen, “Spam Message Filtering Recognition System Based on TensorFlow”, in *2018 3rd International Conference on Mechanical, Control and Computer Engineering (ICMCCE)*, Sep. 2018, S. 564–567. DOI: `10.1109/ICMCCE.2018.00124`.

[57] Sepp Hochreiter und Jürgen Schmidhuber, “Long Short-term Memory”, *Neural computation*, Jg. 9, S. 1735–80, Dez. 1997. DOI: `10.1162/neco.1997.9.8.1735`.

[58] Peter Norvig Stuart Russell, *Artifical Intelligence, A Modern Approach*. Pearson Education Limited, 2016, ISBN: 9780134610993.

[59] Pina Merkert, *Python-Projekte 2020, Von alltagstauglich bis voellig nerdig*. Heise Medien GmbH & Co. KG, 2020.

[60] Ye Zhang und Byron C. Wallace, “A Sensitivity Analysis of (and Practitioners’ Guide to) Convolutional Neural Networks for Sentence Classification”, *CoRR*, Jg. abs/1510.03820, 2015. arXiv: `1510.03820`.

[61] Jason Brownlee, *How to Develop a Multichannel CNN Model for Text Classification*, 2020. Adresse: `https://machinelearningmastery.com/develop-n-gram-multichannel-convolutional-neural-network-sentiment-analysis/`.

[62] Grant Beyleveld, *Multi-ConvNet Sentiment Classifier*, 2019. Adresse: `https://github.com/the-deep-learners/deep-learning-illustrated/blob/master/notebooks/convolutional_sentiment_classifier.ipynb`.

[63] Alon Jacovi, Oren Sar Shalom und Yoav Goldberg, “Understanding Convolutional Neural Networks for Text Classification”, in *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, Brussels, Belgium: Association for Computational Linguistics, Nov. 2018, S. 56–65. DOI: `10.18653/v1/W18-5408`.

[64] ye zhang und byron c. wallace, “a sensitivity analysis of (and practitioners’ guide to) convolutional neural networks for sentence classification”, *corr*, Jg. abs/1510.03820, 2015. arXiv: `1510.03820`.

[65] Cornellius Yudha Wijaya, *Understand Zero-Shot Learning with Python Text Classification Example, Non-Brand Data*, 2023. Adresse: `https://www.nb-data.com/p/understand-zero-shot-learning-with`.

[66] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser und Illia Polosukhin, “Attention Is All You Need”, *CoRR*, Jg. abs/1706.03762, 2017. arXiv: `1706.03762`.

[67] Jacob Devlin, Ming-Wei Chang, Kenton Lee und Kristina Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding”, *arXiv preprint arXiv:1810.04805*, 2018.

[68] RedHat, *Large Language Models (LLMs) im Vergleich zu Small Language Models (SLMs), RedHat*, 2025. Adresse: `https://www.redhat.com/de/topics/ai/llm-vs-slm`.

[69] Zechun Liu, Changsheng Zhao, Igor Fedorov, Bilge Soran, Dhruv Choudhary, Raghuraman Krishnamoorthi, Vikas Chandra, Yuandong Tian und Tijmen Blankevoort, “SpinQuant–LLM quantization with learned rotations”, *arXiv preprint arXiv:2405.16406*, 2024.

[70] Fraunhofer-IAIS, Lamarr-Institute, TU-Dresden, FZ-Jülich, DFKI, Fraunhofer-IIS und Aleph-Alpha, *Teuken-7B-Base & Teuken-7B-Instruct: Towards European LLMs*, 2024. arXiv: 2410.03730 [cs.CL].

[71] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave und Guillaume Lample, *LLaMA: Open and Efficient Foundation Language Models*, 2023. arXiv: 2302.13971 [cs.CL].

[72] VK Cody Bumgardner, Aaron Mullen, Samuel E Armstrong, Caylin Hickey, Victor Marek und Jeff Talbert, "Local Large Language Models for Complex Structured Tasks", *AMIA Summits on Translational Science Proceedings*, Jg. 2024, S. 105, 2024.

[73] pSec Online LLC, *Notice of Claimed Infringement: LLaMA (Large Language Model Meta AI)*, 2023. Adresse: https://github.com/github/dmca/blob/master/2023/03/2023-03-21-meta.md.

[74] Wikipedia, *Llama (language model), Wikipedia*, 2025. Adresse: https://en.wikipedia.org/wiki/Llama_(language_model).

[75] Yukai Zhou, Zhijie Huang, Feiyang Lu, Zhan Qin und Wenjie Wang, "Don't Say No: Jailbreaking LLM by Suppressing Refusal", *arXiv preprint arXiv:2404.16369*, 2024.

[76] David Glukhov, Ilia Shumailov, Yarin Gal, Nicolas Papernot und Vardan Papyan, "LLM Censorship: The Problem and its Limitations", 2024.

[77] Nat McAleese, Rai Michael Pokorny, Juan Felipe Ceron Uribe, Evgenia Nitishinskaya, Maja Trebacz und Jan Leike, *LLM Critics Help Catch LLM Bugs*, 2024. arXiv: 2407.00215 [cs.SE].

[78] Amr Mohamed, Mingmeng Geng, Michalis Vazirgiannis und Guokan Shang, *LLM as a Broken Telephone: Iterative Generation Distorts Information*, 2025. arXiv: 2502.20258 [cs.CL].

[79] Fuzhao Xue, Yao Fu, Wangchunshu Zhou, Zangwei Zheng und Yang You, "To Repeat or Not To Repeat: Insights from Scaling LLM under Token-Crisis", in *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt und S. Levine, Hrsg., Bd. 36, Curran Associates, Inc., 2023, S. 59 304–59 322, ISBN: 9781713871088. Adresse: https://proceedings.neurips.cc/paper_files/paper/2023/file/b9e472cd579c83e2f6aa3459f46aac28-Paper-Conference.pdf.

[80] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra und Christopher Ré, *FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness*, 2022. arXiv: 2205.14135 [cs.LG].

[81] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman und Luke Zettlemoyer, *QLoRA: Efficient Finetuning of Quantized LLMs*, 2023. arXiv: 2305.14314 [cs.LG].

[82] VentureBeat, *Mistral shocks AI community as latest open source model eclipses GPT-3.5 performance*, 2023. Adresse: https://venturebeat.com/ai/mistral-shocks-ai-community-as-latest-open-source-model-eclipses-gpt-3-5-performance/.

[83] UglyRobot, LLC., *Llama 3.1 405B Instruct vs Llama 3.3 70B Instruct, Detailed Performance Feature Comparison*, 2025. Adresse: https://docsbot.ai/models/compare/llama-3-1-405b-instruct/llama-3-3-70b-instruct.

[84] Josef Waples, *Was ist Metas Llama 3.3 70B? Funktionsweise, Anwendungsfälle und mehr, DataCamp*, 2025. Adresse: https://www.datacamp.com/de/blog/llama-3-3-70b.

[85] UglyRobot, LLC., *Llama 2 Chat 70B vs Llama 3.2 11B Vision Instruct, Detailed Performance Feature Comparison*, 2025. Adresse: https://docsbot.ai/models/compare/llama-2-chat-70b/llama-3-2-11b-vision-instruct.

[86] Artificial Analysis, *Mistral Small 3.1 - Intelligence, Performance & Price Analysis, Artificial Analysis*, 2025. Adresse: https://artificialanalysis.ai/models/mistral-small-3-1?models=mistral-small%2Cmistral-8x22b-instruct%2Cmistral-nemo%2Cmixtral-8x7b-instruct.

[87] Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyan Luo, Zhangchi Feng und Yongqiang Ma, “Llamafactory: Unified efficient fine-tuning of 100+ language models”, *arXiv preprint arXiv:2403.13372*, 2024. DOI: 10.18653/v1/2024.acl-demos.38.

[88] Junjie Yin, Jiahao Dong, Yingheng Wang, Christopher De Sa und Volodymyr Kuleshov, “Modulora: Finetuning 3-bit llms on consumer gpus by integrating with modular quantizers”, *arXiv preprint arXiv:2309.16119*, 2023, bitsandbytes. DOI: 10.48550/ARXIV.2309.16119.

[89] Shilpa Devalal und A. Karthikeyan, “LoRa Technology - An Overview”, in *2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA)*, 2018, S. 284–290. DOI: 10.1109/ICECA.2018.8474715.

[90] Kishore Papineni, Salim Roukos, Todd Ward und Wei-Jing Zhu, “Bleu: a Method for Automatic Evaluation of Machine Translation”, in *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, Pierre Isabelle, Eugene Charniak und Dekang Lin, Hrsg., Philadelphia, Pennsylvania, USA: Association for Computational Linguistics, Juli 2002, S. 311–318. DOI: 10.3115/1073083.1073135.

[91] Chin-Yew Lin, “Rouge: A package for automatic evaluation of summaries”, in *Text summarization branches out*, 2004, S. 74–81. DOI: 10.1145/3468264.3468588.

[92] Nahema Marchal, Rachel Xu, Rasmi Elasmar, Iason Gabriel, Beth Goldberg und William Isaac, *Generative AI Misuse: A Taxonomy of Tactics and Insights from Real-World Data*, 2024. arXiv: 2406.13843 [cs.AI].

[93] David Cohen, *Examining Malicious Hugging Face ML Models with Silent Backdoor, JFrog*, 2025. Adresse: https://jfrog.com/blog/data-scientists-targeted-by-malicious-hugging-face-ml-models-with-silent-backdoor/.

[94] Wei Zou, Runpeng Geng, Binghui Wang und Jinyuan Jia, *PoisonedRAG: Knowledge Corruption Attacks to Retrieval-Augmented Generation of Large Language Models*, 2024. arXiv: 2402.07867 [cs.CR].

[95] Anbananthan Pillai Munanday, Norazlianie Sazali, Wan Sharuzi Wan Harun, Kumaran Kadirgama und Ahmad Shahir Jamaludin, “Analysis of convolutional neural networks for facial expression recognition on GPU, TPU and CPU”, *Journal of Advanced Research in Applied Sciences and Engineering Technology*, Jg. 31, Nr. 3, S. 50–67, 2023. DOI: DOI:10.21742/IJCSITE.2020.5.1.04.

[96] Said Salloum, Tarek Gaber, Sunil Vadera und Khaled Shaalan, “A systematic literature review on phishing email detection using natural language processing techniques”, *IEEE Access*, Jg. 10, S. 65 703–65 727, 2022. DOI: 10.1109/ACCESS.2022.3183083.

[97] Joe Chiarella, Jason O Brien, *An Analysis of Spam Filters*, 2003. Adresse: http://www.cs.wpi.edu/~claypool/mqp/spam/.

[98] Mazin Abed Mohammed, Dheyaa Ahmed Ibrahim und Akbal Omran Salman, “Adaptive intelligent learning approach based on visual anti-spam email model for multi-natural language”, *Journal of Intelligent Systems*, Jg. 30, Nr. 1, S. 774–792, 2021. DOI: doi:10.1515/jisys-2021-0045.

[99] venkatesh garnepudi, *spam mails dataset venkatesh garnepudi converted from enron-spam datasets*, 2019. Adresse: https://www.kaggle.com/venky73/spam-mails-dataset.

[100] Kazem Taghandiki, *Building an Effective Email Spam Classification Model with spaCy*, 2023. arXiv: 2303.08792 [cs.AI].

[101] Sridevi Gadde, A. Lakshmanarao und S. Satyanarayana, "SMS Spam Detection using Machine Learning and Deep Learning Techniques", in *2021 7th International Conference on Advanced Computing and Communication Systems (ICACCS)*, Bd. 1, 2021, S. 358–362. DOI: 10.1109/ICACCS51430.2021.9441783.

[102] Cervantes Guevara und Luis Roberto, "Quantum convolutional neural networks for high energy physics", 2022. DOI: 10.1103/PhysRevResearch.4.013231.

[103] Md Tofael Ahmed, Mariam Akter, Md Saifur Rahman, Maqsudur Rahman, Pintu Chandra Paul, Miss Nargis Parvin und Almas Hossain Antar, "Deep neural network based spam email classification using attention mechanisms", *Journal of Intelligent Learning Systems and Applications*, Jg. 15, Nr. 4, S. 144–164, 2023. DOI: 10.4236/jilsa.2023.154010.

[104] Reza Hassanpour, Erdogan Dogdu, Roya Choupani, Onur Goker und Nazli Nazli, "Phishing E-Mail Detection by Using Deep Learning Algorithms", in *Proceedings of the ACMSE 2018 Conference*, Ser. ACMSE '18, Richmond, Kentucky: Association for Computing Machinery, 2018, ISBN: 9781450356961. DOI: 10.1145/3190645.3190719.

[105] Xiaoyong Liao und Guangming Zhou, "The Importance of Token Granularity Matching of Pretrained Word Vectors for Deep Learning-Based Spam Classification", in *2021 3rd International Conference on Natural Language Processing (ICNLP)*, 2021, S. 9–13. DOI: 10.1109/ICNLP52887.2021.00007.

[106] Hwabin Lee, Sua Jeong, Seogyeong Cho und Eunjung Choi, "Visualization technology and deep-learning for multilingual spam message detection", *Electronics*, Jg. 12, Nr. 3, S. 582, 2023. DOI: 10.3390/electronics12030582.

[107] Girija Chetty, Hieu Bui und Matthew White, "Deep Learning Based Spam Detection System", in *2019 International Conference on Machine Learning and Data Engineering (iCMLDE)*, 2019, S. 91–96. DOI: 10.1109/iCMLDE49015.2019.00027.

[108] Jingjing Cai, Jianping Li, Wei Li und Ji Wang, “Deeplearning model used in text classification”, in *2018 15th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP)*, IEEE, 2018, S. 123–126. DOI: `10.1109/ICCWAMTIP.2018.8632592`.

[109] Jyun-Yu Jiang, Mingyang Zhang, Cheng Li, Michael Bendersky, Nadav Golbandi und Marc Najork, “Semantic Text Matching for Long-Form Documents”, in *The World Wide Web Conference*, Ser. WWW ’19, San Francisco, CA, USA: Association for Computing Machinery, 2019, S. 795–806, ISBN: 9781450366748. DOI: `10.1145/3308558.3313707`.

[110] Sanaa Kaddoura, Omar Alfandi und Nadia Dahmani, “A Spam Email Detection Mechanism for English Language Text Emails Using Deep Learning Approach”, in *2020 IEEE 29th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, 2020, S. 193–198. DOI: `10.1109/WETICE49692.2020.00045`.

[111] Isra’a AbdulNabi und Qussai Yaseen, “Spam Email Detection Using Deep Learning Techniques”, *Procedia Computer Science*, Jg. 184, S. 853–858, 2021, The 12th International Conference on Ambient Systems, Networks and Technologies (ANT) / The 4th International Conference on Emerging Data and Industry 4.0 (EDI40) / Affiliated Workshops, ISSN: 1877-0509. DOI: `https://doi.org/10.1016/j.procs.2021.03.107`.

[112] Vijay Srinivas Tida und Sonya Hsu, *Universal Spam Detection using Transfer Learning of BERT Model*, 2022. arXiv: `2202.03480 [cs.CL]`.

[113] Thaer Sahmoud und Mohammad Mikki, “Spam detection using BERT”, *arXiv preprint arXiv:2206.02443*, 2022.

[114] Suhaima Jamal, Hayden Wimmer und Iqbal H. Sarker, “An improved transformer-based model for detecting phishing, spam and ham emails: A large language model approach”, *SECURITY AND PRIVACY*, Jg. 7, Nr. 5, e402, 2024. DOI: `https://doi.org/10.1002/spy2.402`.

[115] Sultan Zavrak und Seyhmus Yilmaz, “Email spam detection using hierarchical attention hybrid deep learning method”, *Expert Systems with Applications*, Jg. 233, S. 120 977, 2023, ISSN: 0957-4174. DOI: `https://doi.org/10.1016/j.eswa.2023.120977`.

[116] Tianrui Liu, Shaojie Li, Yushan Dong, Yuhong Mo und Shuyao He, “Spam detection and classification based on distilbert deep learning algorithm”, *Applied Science and Engineering Journal for Advanced Research*, Jg. 3, Nr. 3, S. 6–10, 2024. DOI: `10.5281/zenodo.11180575`.

[117] Ala'Mahmoud Mohammed Al Zoubi u. a., "Spam Reviews Detection Models in Multilingual Contexts applying Sentiment Analysis, Metaheuristics, and Advanced Word Embedding", 2024.

[118] Athirai A Irissappane, Hanfei Yu, Yankun Shen, Anubha Agrawal und Gray Stanton, "Leveraging GPT-2 for classifying spam reviews with limited labeled data via adversarial training", *arXiv preprint arXiv:2012.13400*, 2020.

[119] Dakota Staples, "A comparison of machine learning algorithms for zero-shot cross-lingual phishing detection", 2023. DOI: `10.1109/PST58708.2023.10320177`.

[120] Maxime Labonne und Sean Moran, *Spam-T5: Benchmarking Large Language Models for Few-Shot Email Spam Detection*, 2023. arXiv: `2304.01238 [cs.CL]`.

[121] OWASP, *OWASP Top 10 for LLM Applications 2025, OWASP Top 10 for LLM Generative AI Security*, 2025. Adresse: `https://genai.owasp.org/resource/owasp-top-10-for-llm-applications-2025/`.

[122] ——, *Whitepapers/Guides LLM Applications Cybersecurity and Governance Checklist v1.1 – English, OWASP Top 10 for LLM Generative AI Security*, 2025. Adresse: `https://genai.owasp.org/resource/llm-applications-cybersecurity-and-governance-checklist-english/`.

[123] Marco Bertenghi, "Angriffe auf große Sprachmodelle", *iX*, Jg. 2025, Nr. 1, S. 42–49, Jan. 2025. Adresse: `ix.de/zt2z`.

[124] ASF Infrabot, *Apache Spamassasin Custom Plugins*, 2020. Adresse: `https://cwiki.apache.org/confluence/display/SPAMASSASSIN/`.

[125] united-domains.de, *Die neuen Domainendungen, Kategorisierung ausgewählter new gTLDs*, 2012. Adresse: `https://blog.united-domains.de/wp-content/uploads/2012/10/Infografik_new_gTLDs_neue_Domainendungen.pdf`.

[126] TitanHQ, *Generic top-level domains (gTLDs) have become a magnet for cybercriminals*, 2020. Adresse: `https://www.titanhq.com/generic-top-level-domains-gtlds-have-become-a-magnet-for-cybercriminals/`.

[127] Holly Esquivel, Aditya Akella und Tatsuya Mori, "On the effectiveness of IP reputation for spam filtering", in *2010 Second International Conference on COMmunication Systems and NETworks (COMSNETS 2010)*, 2010, S. 1–10. DOI: `10.1109/COMSNETS.2010.5431981`.

[128] Azim Khan, *Email Spam Detection with Machine Learning: A Comprehensive Guide, Medium*, 2024. Adresse: `https://medium.com/@azimkhan8018/email-spam-detection-with-machine-learning-a-comprehensive-guide-b65c6936678b`.

[129] Google Developers, *TensorFlow, Build from source*, 2024. Adresse: `https://www.tensorflow.org/install/source#gpu`.

[130] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu und Xiaoqiang Zheng, *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*, Software available from tensorflow.org, 2015. Adresse: `https://www.tensorflow.org/`.

[131] Denis Rothman, *Transformers for Natural Language Processing*. Packt Publishing, 2023, ISBN: 9781803247335.

[132] Thien Nguyen, Lam Nguyen, Phuoc Tran und Huu Nguyen, “Improving Transformer-Based Neural Machine Translation with Prior Alignments”, *Complexity*, Jg. 2021, Nr. 1, S. 5 515 407, 2021, Trax. DOI: `10.1155/2021/5515407`.

[133] Quan Hui, Bing Liu, Jianchun Fan, Zeheng Peng und Yuguo Wu, “Application of Standard Intelligent Translation Technology Based on Open Source Translation System of Oil and Gas Pipeline Network Standard Terminology Database”, in *2024 5th International Seminar on Artificial Intelligence, Networking and Information Technology (AINIT)*, Argostranslate, IEEE, 2024, S. 164–170. DOI: `10.1109/AINIT61980.2024.10581457`.

[134] Christiane Fellbaum, *WordNet: An Electronic Lexical Database*. Bradford Books, 1998. Adresse: `https://mitpress.mit.edu/9780262561167/`.

[135] NVIDIA Corporation, *Support Matrix, GPU, CUDA Toolkit, and CUDA Driver Requirements*, 2024. Adresse: `https://docs.nvidia.com/deeplearning/cudnn/latest/reference/support-matrix.html`.

[136] NVIDIA, *CUDA Python 12.3.0 Release notes*, 2023. Adresse: `https://nvidia.github.io/cuda-python/cuda-bindings/latest/release/12.3.0-notes.html`.

[137] Alaa El-Halees, “Filtering Spam E-Mail from Mixed Arabic and English Messages: A Comparison of Machine Learning Techniques.”, *International Arab Journal of Information Technology (IAJIT)*, Jg. 6, Nr. 1, 2009.

[138] Ollama, *ollama, library*, 2025. Adresse: `https://ollama.com/library`.

[139] semrush, *Top 6 huggingface.co, Alternatives Competitors*, 2025. Adresse: `https://www.semrush.com/website/huggingface.co/competitors/`.

[140] Hugging Face, *Models overview*, 2025. Adresse: `https://huggingface.co/models?pipeline_tag=text-classification&sort=trending`.

[141] Shunichi Amari, “A Theory of Adaptive Pattern Classifiers”, *IEEE Transactions on Electronic Computers*, Jg. EC-16, Nr. 3, S. 299–307, 1967. DOI: `10.1109/PGEC.1967.264666`.

[142] Eliyahu Kiperwasser und Yoav Goldberg, “Simple and accurate dependency parsing using bidirectional LSTM feature representations”, *Transactions of the Association for Computational Linguistics*, Jg. 4, S. 313–327, 2016. DOI: `10.1162/tacl_a_00101`.

[143] Kunihiko Fukushima, “Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position”, *Biological cybernetics*, Jg. 36, Nr. 4, S. 193–202, 1980. DOI: `10.1007/BF00344251`.

[144] Dipanjan Sarkar, *Text Analytics with Python, A Practioner’s Guide to Natural Language Processing*. Apress, 2016, ISBN: 9781484243534.

[145] Dan Cireşan, Ueli Meier und Juergen Schmidhuber, *Multi-column Deep Neural Networks for Image Classification*, 2012. arXiv: `1202.2745 [cs.CV]`.

[146] Mike Lewis, “Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension”, *arXiv preprint arXiv:1910.13461*, 2019.

[147] Chaofan Li, Zheng Liu, Shitao Xiao und Yingxia Shao, “Making large language models a better foundation for dense retrieval”, *arXiv preprint arXiv:2312.15503*, 2023.

[148] Yuanzhi Li, Sébastien Bubeck, Ronen Eldan, Allie Del Giorno, Suriya Gunasekar und Yin Tat Lee, “Textbooks Are All You Need II: **phi-1.5** technical report”, *arXiv preprint arXiv:2309.05463*, 2023.

[149] Marah Abdin, Jyoti Aneja, Hany Awadalla, Ahmed Awadallah, Ammar Ahmad Awan, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Jianmin Bao, Harkirat Behl u. a., “Phi-3 technical report: A highly capable language model locally on your phone”, *arXiv preprint arXiv:2404.14219*, 2024.

[150] Marah Abdin, Jyoti Aneja, Harkirat Behl, Sébastien Bubeck, Ronen Eldan, Suriya Gunasekar, Michael Harrison, Russell J Hewett, Mojan Javaheripi, Piero Kauffmann u. a., “Phi-4 technical report”, *arXiv preprint arXiv:2412.08905*, 2024.

[151] Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Gabriel Martın Blázquez, Guilherme Penedo, Lewis Tunstall, Andrés Marafioti, Hynek Kydlıček, Agustın Piqueres Lajarın, Vaibhav Srivastav u. a., “SmolLM2: When Smol Goes Big–Data-Centric Training of a Small Language Model”, *arXiv preprint arXiv:2502.02737*, 2025.

[152] DeepSeek-AI-Team, *DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning*, 2025. arXiv: 2501.12948 [cs.CL].

[153] Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier u. a., “Mistral 7B”, *arXiv preprint arXiv:2310.06825*, 2023.

[154] Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix und William El Sayed, *Mixtral of Experts*, 2024. arXiv: 2401.04088 [cs.LG].

[155] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever u. a., “Language models are unsupervised multitask learners”, *OpenAI blog*, Jg. 1, Nr. 8, S. 9, 2019. Adresse: https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf.

[156] Nikolay Bogoychev, Jelmer Van der Linde und Kenneth Heafield, “TranslateLocally: Blazing-fast translation running on the local CPU”, *arXiv preprint arXiv:2109.10194*, 2021.

[157] P.J. Finlay, *Argos-Translate: Open-source offline translation library written in Python, GitHub*, 2025. Adresse: https://github.com/argosopentech/argos-translate.

[158] Yuli Vasiliev, *Natural language processing with Python and spaCy: A practical introduction*. No Starch Press, 2020, ISBN: 9781718500525.

[159] Denis Rothman, *Transformers for Natural Language Processing: Build, train, and fine-tune deep neural network architectures for NLP with Python, Hugging Face, and OpenAI's GPT-3, ChatGPT, and GPT-4*. Packt Publishing Ltd, 2022, ISBN: 9781803247335.

[160] Vaclav Smil, *Energy : a beginner's guide*. Oxford : Oneworld, 2009, ISBN: 978-1786071330.

[161] Dr. Karl-Heinz Best, *Wortlängen verschiedener Sprachen im Vergleich*, 2021. Adresse: https://de.wikipedia.org/wiki/Wortl%C3%A4nge#Wortl%C3%A4ngen_verschiedener_Sprachen_im_Vergleich.

[162] Wahiba Ben Abdessalem Karaa und Nidhal Gribâa, "Information Retrieval with Porter Stemmer: A New Version for English", in *Advances in Computational Science, Engineering and Information Technology*, Dhinaharan Nagamalai, Ashok Kumar und Annamalai Annamalai, Hrsg., Heidelberg: Springer International Publishing, 2013, S. 243–254, ISBN: 978-3-319-00951-3. DOI: 10.1007/978-3-319-00951-3_24.

[163] Leonie Weissweiler und Alexander Fraser, "Developing a Stemmer for German Based on a Comparative Analysis of Publicly Available Stemmers", in *Language Technologies for the Challenges of the Digital Age*, Georg Rehm und Thierry Declerck, Hrsg., Cham: Springer International Publishing, 2018, S. 81–94, ISBN: 978-3-319-73706-5.

[164] Divya Khyani, BS Siddhartha, NM Niveditha und BM Divya, "An interpretation of lemmatization and stemming in natural language processing", *Journal of University of Shanghai for Science and Technology*, Jg. 22, Nr. 10, S. 350–357, 2021.

[165] Joel Nothman, Nicky Ringland, Will Radford, Tara Murphy und James R. Curran, "Learning multilingual named entity recognition from Wikipedia", *Artificial Intelligence*, Jg. 194, S. 151–175, 2013, Artificial Intelligence, Wikipedia and Semi-Structured Resources, ISSN: 0004-3702. DOI: https://doi.org/10.1016/j.artint.2012.03.006.

[166] Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart und Alexander Rush, "OpenNMT: Open-Source Toolkit for Neural Machine Translation", in *Proceedings of ACL 2017, System Demonstrations*, Vancouver, Canada: Association for Computational Linguistics, Juli 2017, S. 67–72. arXiv: 1701.02810. Adresse: https://www.aclweb.org/anthology/P17-4012.

[167] Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand u. a., "Mixtral of experts", *arXiv preprint arXiv:2401.04088*, 2024.

[168] unsloth, *unsloth/Llama-3.2-1B Hugging Face*, 2024. Adresse: https://huggingface.co/unsloth/Llama-3.2-1B.

[169] C. S. Wallace und D. M. Boulton, “An Information Measure for Classification”, *The Computer Journal*, Jg. 11, Nr. 2, S. 185–194, Aug. 1968, ISSN: 0010-4620. DOI: 10.1093/comjnl/11.2.185.

[170] Future of Life Institute, *Implementation Timeline, EU Artificial Intelligence Act*, 2025. Adresse: https://artificialintelligenceact.eu/implementation-timeline/.

[171] Nicolas Patry, *huggingface/safetensors: Simple, safe way to store and distribute tensors, GitHub*, 2025. Adresse: https://github.com/HuggingFace/safetensors.

[172] ExplosionAI GmbH, *Install spaCy, spaCy Usage Documentation*, 2025. Adresse: https://spacy.io/usage.

[173] Dirk Helbing, Bruno S Frey, Gerd Gigerenzer, Ernst Hafen, Michael Hagner, Yvonne Hofstetter, Jeroen Van Den Hoven, Roberto V Zicari und Andrej Zwitter, “Will democracy survive big data and artificial intelligence?”, *Towards digital enlightenment: Essays on the dark and light sides of the digital revolution*, S. 73–98, 2019. DOI: 10.1007/978-3-319-90869-4_7.

[174] Linnea Olby und Isabel Thomander, *A Step Toward GDPR Compliance: Processing of Personal Data in Email*, 2018.

[175] Future of Life Institute, *High-level summary of the AI Act, EU Artificial Intelligence Act*, 2025. Adresse: https://artificialintelligenceact.eu/high-level-summary/.

[176] Generaldirektion Kommunikationsnetze, Inhalte und Technologien, *KI-Gesetz, Gestaltung der digitalen Zukunft Europas*, 2025. Adresse: https://digital-strategy.ec.europa.eu/de/policies/regulatory-framework-ai.

[177] RUNDFUNK UND TELEKOM REGULIERUNGS-GMBH, *Risikostufen von KI-Systemen, KI-Servicestelle*, 2025. Adresse: https://www.rtr.at/rtr/service/ki-servicestelle/ai-act/risikostufen_ki-systeme.de.html.

[178] David Patterson, Joseph Gonzalez, Urs Hölzle, Q Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David So, Maud Texier und Jeffrey Dean, “The Carbon Footprint of Machine Learning Training Will Plateau, Then Shrink, Apr. 2022”, *URL http://arxiv. org/abs/2204.05149*, 2022.

[179] Yifan Sun, Nicolas Bohm Agostini, Shi Dong und David Kaeli, “Summarizing CPU and GPU design trends with product data”, *arXiv preprint arXiv:1911.11313*, 2019.

[180] Batuhan Hangün und Önder Eyecioğlu, “Performance comparison between OpenCV built in CPU and GPU functions on image processing operations”, *International Journal of Engineering Science and Application*, Jg. 1, Nr. 2, S. 34–41, 2017. DOI: `10.48550/arXiv.1906.08819`.

[181] Vsevolod Stakhov, *Enhancing Email Spam Detection with LLMs – Practical Experience with Rspamd and GPT*, 2025. Adresse: `https://fosdem.org/2025/schedule/event/fosdem-2025-5114-enhancing-email-spam-detection-with-llms-practical-experience-with-rspamd-and-gpt/`.

[182] Wikipedia contributors, *Human brain — Wikipedia, The Free Encyclopedia*, 2025. Adresse: `https://en.wikipedia.org/w/index.php?title=Human_brain&oldid=1279715436`.

[183] Leela S Karumbunathan, “Nvidia jetson agx orin series”, *Online at https://www. nvidia. com/content/dam/en-zz/Solutions/gtcf21/jetson-orin/nvidia-jetson-agx-orin-technical-brief. pdf*, 2022.

[184] Simran Gibson, Biju Issac, Li Zhang und Seibu Mary Jacob, “Detecting spam email with machine learning optimized with bio-inspired metaheuristic algorithms”, *Ieee Access*, Jg. 8, S. 187 914–187 932, 2020. DOI: `10.1109/ACCESS.2020.3030751`.

[185] Yongchao Yuan, Shaoqing Lv, Zhiqiang Bao und Kai Li, “A Joint Model for Text Classification with BERT-BiLSTM and GCN”, in *Proceedings of the 2022 5th International Conference on Artificial Intelligence and Pattern Recognition*, 2022, S. 180–186. DOI: `10.1145/3573942.3573970`.

[186] Duc Nguyen Huu, *LLMs-for-Text-Classification/phi3*$_{c}lassifier.ipynbatmain$*, ducnh279/LLMs-for-Text-Classification · GitHub*, 2024. Adresse: `https://github.com/ducnh279/LLMs-for-Text-Classification/blob/main/phi3_classifier.ipynb`.

A. Anhang Jupyter-Notebooks und Skripte

Eine Übesicht ist im Kapitel 4.4.2 zu finden.

A.1. Archivierung, Umwandlung in CSV

A.1.1. CheckAntispam.py

```
1  #!/usr/bin/python3
2  # Check_antispam getestet auf Debian 11 mit Python3 und
   ↪  tensorflow 2.6.0
3  # Als root: apt install python3-keras pip
4  # Als User: pip3 install tensorflow pandas
5
6  import email
7  import sys
8  import subprocess
9  import logging
10
11 import tensorflow as tf
12 from tensorflow.keras.preprocessing.text import Tokenizer
13 from tensorflow.keras.preprocessing.sequence import pad_sequences
14 from tensorflow.keras.models import Sequential
15 from tensorflow.keras.layers import Embedding,
16         GlobalAveragePooling1D,  Flatten, Dense, Dropout,
         ↪  Embedding, LSTM,
17         SpatialDropout1D, Bidirectional, SpatialDropout1D,
         ↪  Conv1D, GlobalMaxPooling1D
18 from keras.layers import Input, concatenate # Multi-ConvNet
19 from keras.models import Model               # Multi-ConvNet
20 from tensorflow import keras                 # ist noetig
21 import pickle        # Load Save Tokenizer
```

```
22  import pandas as pd
23  import os               # Reading and writing from Disk
24  
25  import quopri           # for decode
26  from email.header import decode_header
27  from bs4 import BeautifulSoup                  # html2text pip3
     ↪  install  BeautifulSoup4 lxml
28  import sys              # cmd input parameter
29  import difflib          # compare plain 2 html
30  import re               # for removing URLs
31  import fasttext         # Language Detection pip3
32  
33  version     = "40"
34  pathHome    = "/home/ralph/"
35  MAX_LEN = 512 #300      # pad_sequencs parameter, only look for X
     ↪  words in a sentence (80% )
36  trunc_type = "post" # pad_sequences parameter
37  padding_type = "post" # pad_sequences parameter
38  oov_tok = "<OOV>" # out of vocabulary token
39  VOCAB_SIZE = 32768 #15626   # need a fix size !
40  
41  verbose = False
42  if(len(sys.argv) > 1):
43      try:
44          arg = sys.argv[1:]
45          for a in arg:
46              if(a == "-v"):
47                  print("Verbose Output")
48                  verbose = True
49      except IndexError:
50          raise SystemExit(f"Usage: {sys.argv[0]} <-v> <-[h|s]>
             ↪  <path>")
51      if(verbose):
52          print(arg[::-1])
53  sender      = sys.argv[1]   # is needed for Sendmail call
54  recipient   = sys.argv[2]   # is needed for Sendmail call
55  
56  sys.stdin = sys.stdin.detach()
57  message = email.message_from_binary_file(sys.stdin)
```

```
58
59  LOG_FORMAT = "%(levelname)s %(asctime)s - %(message)s"
60  logging.basicConfig(filename = pathHome + sys.argv[0] + ".log",
    ↪  format = LOG_FORMAT, filemode = 'a', level = logging.INFO)
61  logger = logging.getLogger()
62
63
64  isSPAM = True
65  mailtext = ""
66
67  if(verbose):
68      print("-----------------------------------------------    --
        ↪  -    -   ")
69  fromExtern = True
70  if 'Received' in message:
71      res =  message['Received']
72      rec = res[5:23]
73      mx = "mx.services.ama.at"
74      i = 0
75      while (i < 18):
76          if rec[i] == mx[i]:
77              next
78          else:
79              fromExtern = False
80              break
81          i = i + 1
82  if fromExtern :          # Dedection with Received, some are not
    ↪  from extern...
83      if 'Subject' in message:
84          try:
85              mailtext = ""
86              sub =  message['Subject']
87              if(verbose):
88                  print(sub)
89              ext = ascii(sub)[0:20]
90              if(verbose):
91                  print(ext)
92              spamtext = ascii("*****SPAM*****")
93              isSPAM = True
```

```
94                 isExtern = True
95                 i = 0
96                 if(len(ascii(sub))>=14):
97                     while (i < 14):
98                         if ext[i] == spamtext[i]:
99                             next
100                        else:
101                            isSPAM = False
102                            break
103                        i = i + 1
104                    if isSPAM == True :
105                        sub = sub[20:]  # cut Text
106                    else:
107                        i = 0
108                        ext = ascii(sub)[0:5]
109                        extText = ascii("[EXT]")
110                        while (i < 5):
111                            if ext[i] == extText[i]:
112                                next
113                            else:
114                                isExtern = False
115                                break
116                            i = i + 1
117                    if isExtern == True :
118                        if 'message-id' in message:
119                            if(verbose):
120                                print('Message-ID:',
                                    ↪ message['message-id'])
121                        sub = sub[6:]
122                        sub = sub.replace('\n','')  # newline und
                            ↪ carrage retrun entfernen
123                        sub = sub.replace('\r','')
124                        decHeader = decode_header(sub)
125                        utf8 = quopri.decodestring(sub)  # Decode
                            ↪ quoted-printable to raw bytes.
126                        sub = utf8.decode('utf-8')    # Decode bytes
                            ↪ to tex
127                        utfText = "=?utf-8?"
128                        isUTF = True
```

```
129                     i = 0
130                     if(len(sub)>=10):
131                         while (i < 8):
132                             if sub[i].lower() == utfText[i]:
133                                 next
134                             else:
135                                 isUTF = False
136                                 break
137                             i = i + 1
138                         if(isUTF == True):
139                             sub = sub[10:len(sub)-1]
140                             sub = sub.replace('_',' ')
141                             x = sub.lower().find(utfText)
142                             if(x>0):
143                                 sub = sub[0:x] + sub[x+10:]
144                                 x = sub.lower().find(utfText)
145                                 if(x>0):
146                                     sub = sub[0:x] + sub[x+10:]
147                                     x = sub.lower().find(utfText)
148                                     if(x>0):
149                                         sub = sub[0:x] +
                                         ↪  sub[x+10:]
150                                         x = sub.lower().find(
                                         ↪  utfText)
151                                         if(x>0):
152                                             sub = sub[0:x] +
                                             ↪  sub[x+10:]
153                     if(verbose):
154                             print(' Subject: ', sub)
155                     cntPlain = 0     # Count of found plain Parts,
                     ↪  because if its a SpamAssasin Mail the
                     ↪  first should be skipped
156                     cntHtml = 0
157                     text = ""
158                     for part in message.walk():
159                         if(verbose):
160                             print(" get_content_type:\t"+
                             ↪  part.get_content_type())
161                         charset = "utf-8"            # default
                         ↪  charset
```

```
162                         for char in message.get_charsets():
163                             if(char is not None):
164                                 charset = char
165                                 break               # End for
                                 ↪   loop if found
166                         if(part.get_content_type() ==
                         ↪   "text/plain"):
167                             cntPlain = cntPlain + 1
168                             if(cntPlain == 1 and isSPAM == True):
                             ↪   # I do not want to use the
                             ↪   SPAMassin Text for Training
169                                 if(verbose):
170                                     print("
                                     ↪   -----------SPAMassasin-----")
171                             else:
172                                 if(verbose):
173                                     print("
                                     ↪   -----------PLAIN---------")
174                                 text =
                                 ↪   part.get_payload(decode=True)
175                                 text = text.decode(charset)
176                                 if(verbose):
177                                     print(len(text))
178                                 if(len(text) > 32000):
179                                     text = text[0:31999]
180                                     if(verbose):
181                                         print(" tolong->Cut")
182                                 text= text.replace('\\n',' ')    #
                                 ↪   newline und carrage retrun
                                 ↪   entfernen
183                                 text= text.replace('\n',' ')
184                                 text= text.replace('\\r',' ')
185                                 text= text.replace('\r',' ')
186                                 text= text.replace('\\t',' ')
187                                 text= text.replace('\t',' ')
188                                 text = re.sub('http[s]?://(?:[a-
                                 ↪   zA-Z]|[0-9]|[$-
                                 ↪   _@.&+]|[!*\(\),]|(?:%[0-9a-
                                 ↪   fA-F][0-9a-fA-F]))+', '',
                                 ↪   text, flags=re.MULTILINE)
```

```
                                text=text.replace(';',' ')
                                text=text.replace('"',' ')
                                text=text.replace('`',' ')
                                text=text.replace("'",' ')
                                text=text.replace(':',' ')
                                text= text.replace('      ',' ')
                                text= text.replace('    ',' ')
                                text= text.replace('   ',' ')
                                text= text.replace('  ',' ')
                                text= text.replace('_','')
                                text= text.replace('-','')
                                text= text.replace('|','')
                                if(len(text) > 200):
                                    if(verbose):
                                        print("  ", text[0:200])
                                else:
                                    if(verbose):
                                        print("  ", text)
                                if(len(text) >= 10):
                                    mailtext = mailtext + text
                        if(part.get_content_type() ==
                         ↪  "text/html"):
                            cntHtml = cntHtml + 1
                            if(verbose):
                                print("
                                 ↪  ------------HTML---------")
                            htmlori =
                             ↪  part.get_payload(decode=True)
                            soup =
                             ↪  BeautifulSoup(htmlori,features=
                             ↪  "lxml")
                            html = soup.get_text()
                            if(verbose):
                                print(len(htmlori))
                            if (len(htmlori) > 2):      # even
                             ↪  when empty its 2...
                                if(len(html) > 32000):
                                    html = html[0:31999]
                                    if(verbose):
```

```
222                         print(" tolong->Cut")
223                     html = html.replace('\\n',' ')   #
                         ↪ newline und carrage retrun
                         ↪ entfernen
224                     html = html.replace('\n',' ')
225                     html = html.replace('\\r',' ')
226                     html = html.replace('\r',' ')
227                     html = html.replace('\\t',' ')
228                     html = html.replace('\t',' ')
229                     if(verbose):
230                         print(" vorSub")
231                     html = re.sub('http[s]?://(?:[a-
                         ↪ zA-Z]|[0-9]|[$-
                         ↪ _@.&+]|[!*\(\),]|(?:%[0-9a-
                         ↪ fA-F][0-9a-fA-F]))+', '',
                         ↪ html, flags=re.MULTILINE)
232                     if(verbose):
233                         print(" nachSub")
234                     html = html.replace(',',' ')
235                     html = html.replace(';',' ')
236                     html = html.replace('"',' ')
237                     html = html.replace('`',' ')
238                     html = html.replace("'",' ')
239                     html = html.replace(':',' ')
240                     if(verbose):
241                         print(" vorLeerz")
242                     html = html.replace('      ',' ')
243                     html = html.replace('    ',' ')
244                     html = html.replace('   ',' ')
245                     html = html.replace('  ',' ')
246                     html = html.replace('_','')
247                     html = html.replace('-','')
248                     html = html.replace('|','')
249                     if(verbose):
250                         print(" nachReplace")
251                     r = 0       #Ratio default value
252                     if(cntPlain > 0):
253                         s =
                             ↪ difflib.SequenceMatcher(None,
                             ↪ text, html, True)
```

```
254                                 a = len(ascii(text))
255                                 b = len(ascii(html))
256                                 r = s.ratio()
257                                 if(verbose):
258                                     print(" len(a/b), ratio:
                                     ↪  ",a,"/",b,", ",r)
259                             if(len(html) > 200):
260                                 if(verbose):
261                                     print(" ", html[0:200])
262                             else:
263                                 if(verbose):
264                                     print(" ", html)
265                             if(r < 0.6):
266                                 if(len(html) >= 10):     # to
                                 ↪  short no real value
267                                     mailtext = mailtext +
                                     ↪  html
268                     if(verbose):
269                         print(" cntPlain/Html: ",cntPlain,
                         ↪  "/", cntHtml)
270                     if(cntHtml + cntPlain > 1):     # End if
                     ↪  saved part is bigger then 1...
271                         if(verbose):
272                             print(" mooooooooooooore then 1
                             ↪  saved")
273                         break
274         except UnicodeDecodeError:
275                             print(" UnicodeDecodeError")
276 if(verbose):
277     print(mailtext[0:150])
278     logger.info(mailtext[0:150])
279 with open('modelRun/model'+ version +'.pickle', 'rb') as handle:
280     bestList = pickle.load(handle)
281 if(verbose):
282     for best in bestList:
283         print(best)
284 mailLen = len(mailtext)          # get length og mail, for
    ↪  choosing the right model
285 if(verbose):
```

```
286         print(str(mailLen))
287     if(mailLen <= 1500):
288         msglength = "short"
289         if(verbose):
290                 print("short")
291     elif(mailLen < 3000):
292         msglength = "middle"
293         if(verbose):
294                 print("middle")
295     else:
296         msglength = "long"
297         if(verbose):
298                 print("long")
299     PRETRAINED_MODEL_PATH = 'lid.176.bin'        # fastText
300     modelLang = fasttext.load_model(PRETRAINED_MODEL_PATH)
301     lang = modelLang.predict(mailtext)[0][0]     # dedect the language
302     langAll = modelLang.predict(mailtext)
303     lang = langAll[0][0]
304     langP = langAll[1][0]
305     if(verbose):
306         print(lang)
307         print(langP)
308     tokenDir= pathHome + "model_output/" + msglength + version + lang
         ↪  + "/"
309     tokenizer = None
310     with open(tokenDir+ 'tokenizer.pickle', 'rb') as handle:
311         tokenizer = pickle.load(handle)
312     EMBEDING_DIM = 32        # deepends on Vocab Size  2 power 32
313     DROP_VALUE = 0.2         # dropout
314     trunc_type = "post"      # pad_sequences parameter
315     padding_type = "post"    # pad_sequences parameter
316     oov_tok = "<OOV>"        # out of vocabulary token
317
318 def predict_spam(mod, predict_msg, tokenizer, msglength, version,
         ↪  lang): # callc probabillity for spam
319     DROP_VALUE = drop_embed = dropout = 0.2 # dropout
320     N_DENSE = 24
321     epochs = 30
322     batch_size = 128
```

```
323     N_LTSM = 20
324     pad_type = trunc_type = 'pre'    # vector-space embedding
325     optim = keras.optimizers.Adam(learning_rate=0.00001)  #
         ↪  optimiser to avoid local minima
326     modelDir = "modelRun/" + mod + msglength + version + lang
327 
328     if(mod == "dense0"): #Dense sentiment model architecture
329         model = Sequential()
330         model.add(Embedding(VOCAB_SIZE, EMBEDING_DIM,
             ↪  input_length=MAX_LEN))
331         model.add(GlobalAveragePooling1D())
332         model.add(Dense(N_DENSE, activation='relu'))
333         model.add(Dropout(DROP_VALUE))
334         model.add(Dense(1, activation='sigmoid'))
335     elif(mod == "ltsm"): #LSTM Spam detection architecture
336         model = Sequential()
337         model.add(Embedding(VOCAB_SIZE, EMBEDING_DIM,
             ↪  input_length=MAX_LEN))
338         model.add(LSTM(N_LTSM, dropout=DROP_VALUE,
             ↪  return_sequences=True))
339         model.add(LSTM(N_LTSM, dropout=DROP_VALUE,
             ↪  return_sequences=True))
340         model.add(GlobalAveragePooling1D())
341         model.add(Dense(1, activation='sigmoid'))
342     elif(mod == "bi-ltsm"):# Biderectional LSTM Spam detection
         ↪  architecture
343         model = Sequential()
344         model.add(Embedding(VOCAB_SIZE, EMBEDING_DIM,
             ↪  input_length=MAX_LEN))
345         model.add(Bidirectional(LSTM(N_LTSM, dropout=DROP_VALUE,
             ↪  return_sequences=True)))
346         model.add(GlobalAveragePooling1D())
347         model.add(Dense(1, activation='sigmoid'))
348     elif(mod == "conv3"):      # deep-learning-illustrated-
         ↪  master/notebooks/convolutional_sentiment_classifier.ipynb
349         max_review_length = 400
350         N_CONV = 256 # filters, a.k.a. kernels
351         k_conv = 3 # kernel length
352         N_DENSE = 256
```

```
353            model = Sequential()
354            model.add(Embedding(VOCAB_SIZE, EMBEDING_DIM,
                ↪ input_length=MAX_LEN))
355            model.add(SpatialDropout1D(DROP_VALUE))
356            model.add(Conv1D(N_CONV, k_conv, activation='relu'))
357            model.add(Conv1D(N_CONV, k_conv, activation='relu'))
358            model.add(GlobalMaxPooling1D())
359            model.add(Dense(N_DENSE, activation='relu'))
360            model.add(Dropout(DROP_VALUE))
361            model.add(Dense(1, activation='sigmoid'))
362        elif(mod == "multiconv"):
363            n_conv_1 = n_conv_2 = n_conv_3 = 256    # convolutional
                ↪ layer architecture
364            k_conv_1 = 3
365            k_conv_2 = 2
366            k_conv_3 = 4
367            n_dense = 256   # dense layer architecture
368            input_layer = Input(shape=(MAX_LEN,), dtype='int16',
                ↪ name='input')
369            embedding_layer = Embedding(VOCAB_SIZE, EMBEDING_DIM,
                ↪ name='embedding')(input_layer)    # embedding
370            drop_embed_layer = SpatialDropout1D(drop_embed,
                ↪ name='drop_embed')(embedding_layer)
371            conv_1 = Conv1D(n_conv_1, k_conv_1, activation='relu',
                ↪ name='conv_1')(drop_embed_layer) # three parallel
                ↪ convolutional streams:
372            maxp_1 = GlobalMaxPooling1D(name='maxp_1')(conv_1)
373            conv_2 = Conv1D(n_conv_2, k_conv_2, activation='relu',
                ↪ name='conv_2')(drop_embed_layer)
374            maxp_2 = GlobalMaxPooling1D(name='maxp_2')(conv_2)
375            conv_3 = Conv1D(n_conv_3, k_conv_3, activation='relu',
                ↪ name='conv_3')(drop_embed_layer)
376            maxp_3 = GlobalMaxPooling1D(name='maxp_3')(conv_3)
377            concat = concatenate([maxp_1, maxp_2, maxp_3])   #
                ↪ concatenate the activations from the three streams
378            dense_layer = Dense(n_dense, activation='relu',
                ↪ name='dense')(concat) # dense hidden layers
379            drop_dense_layer = Dropout(dropout,
                ↪ name='drop_dense')(dense_layer)
```

```
380            dense_2 = Dense(int(n_dense/4), activation='relu',
                ↪  name='dense_2')(drop_dense_layer)
381            dropout_2 = Dropout(dropout,
                ↪  name='drop_dense_2')(dense_2)
382            predictions = Dense(1, activation='sigmoid',
                ↪  name='output')(dropout_2) # sigmoid output layer
383            model = Model(input_layer, predictions) # create model
384        elif(mod == "multiconv5"):
385            n_conv = 768     # convolutional layer architecture:
386            k_conv_1 = 1
387            k_conv_2 = 2
388            k_conv_3 = 3
389            k_conv_4 = 5
390            k_conv_5 = 7
391            k_conv_6 = 9
392            n_dense = 512    # dense layer architecture:
393            input_layer = Input(shape=(MAX_LEN,), dtype='int16',
                ↪  name='input')
394            embedding_layer = Embedding(VOCAB_SIZE, EMBEDING_DIM,
                ↪  name='embedding')(input_layer)     # embedding:
395            drop_embed_layer = SpatialDropout1D(drop_embed,
                ↪  name='drop_embed')(embedding_layer)
396            conv_1 = Conv1D(n_conv, k_conv_1, activation='relu',
                ↪  name='conv_1')(drop_embed_layer)    # three parallel
                ↪  convolutional streams:
397            maxp_1 = GlobalMaxPooling1D(name='maxp_1')(conv_1)
398            conv_2 = Conv1D(n_conv, k_conv_2, activation='relu',
                ↪  name='conv_2')(drop_embed_layer)
399            maxp_2 = GlobalMaxPooling1D(name='maxp_2')(conv_2)
400            conv_3 = Conv1D(n_conv, k_conv_3, activation='relu',
                ↪  name='conv_3')(drop_embed_layer)
401            maxp_3 = GlobalMaxPooling1D(name='maxp_3')(conv_3)
402            conv_4 = Conv1D(n_conv, k_conv_4, activation='relu',
                ↪  name='conv_4')(drop_embed_layer)
403            maxp_4 = GlobalMaxPooling1D(name='maxp_4')(conv_4)
404            conv_5 = Conv1D(n_conv, k_conv_5, activation='relu',
                ↪  name='conv_5')(drop_embed_layer)
405            maxp_5 = GlobalMaxPooling1D(name='maxp_5')(conv_5)
406            conv_6 = Conv1D(n_conv, k_conv_6, activation='relu',
                ↪  name='conv_6')(drop_embed_layer)
```

```
407             maxp_6 = GlobalMaxPooling1D(name='maxp_6')(conv_6)
408             concat12 = concatenate([maxp_1, maxp_2])
409             concat34 = concatenate([maxp_3, maxp_4])
410             concat56 = concatenate([maxp_5, maxp_6])
411             concat = concatenate([concat12, concat34, concat56])
412             dense_layer = Dense(n_dense, activation='relu',
                 ↪  name='dense')(concat)    # dense hidden layers:
413             drop_dense_layer = Dropout(dropout,
                 ↪  name='drop_dense')(dense_layer)
414             dense_2 = Dense(int(n_dense/4), activation='relu',
                 ↪  name='dense_2')(drop_dense_layer)
415             dropout_2 = Dropout(dropout,
                 ↪  name='drop_dense_2')(dense_2)
416             predictions = Dense(1, activation='sigmoid',
                 ↪  name='output')(dropout_2)   # sigmoid output layer:
417             model = Model(input_layer, predictions) # create model
418
419         model.load_weights(modelDir + ".hdf5")
420         new_seq = tokenizer.texts_to_sequences(predict_msg)
421         padded = pad_sequences(new_seq, maxlen =MAX_LEN, padding =
             ↪  padding_type, truncating=trunc_type)
422         return (model.predict(padded))# returns the probabillity for
             ↪  spam
423
424     mail_dict = {'message': [mailtext]}
425     mail_pd = pd.DataFrame(mail_dict)
426     if(msglength == "short"):
427         if(lang == "__label__de"):
428             mod = bestList[0][1]
429         if(lang == "__label__en"):
430             mod = bestList[1][1]
431         if(lang == "__label__somtehingElse"):
432             mod = bestList[2][1]
433     elif(msglength == "middle"):
434         if(lang == "__label__de"):
435             mod = bestList[3][1]
436         if(lang == "__label__en"):
437             mod = bestList[4][1]
438         if(lang == "__label__somtehingElse"):
```

```
439            mod = bestList[5][1]
440    elif(msglength == "long"):
441        if(lang == "__label__de"):
442            mod = bestList[6][1]
443        if(lang == "__label__en"):
444            mod = bestList[7][1]
445        if(lang == "__label__somtehingElse"):
446            mod = bestList[8][1]
447    else:
448            mod = "conv3"
449            print("ERROR should never be reached")
450
451    y_hat =(predict_spam(mod, mail_pd['message'], tokenizer,
        ↪  msglength, version, lang))
452
453    logtext = 'Error'
454    if (y_hat > .5):    # SPAM
455        logtext = "
          ↪  SPAM;"+str(y_hat[0][0])+";"+msglength+";"+version+";"+lang+
          ↪  ";"+str(langP)+";"+mod+";"+sender+";"+recipient+";"
456        message.add_header('X-KISPAM: ' + logtext, 'YES')        # for
          ↪  Debug
457        if(verbose):
458            print('X-KISPAM: ' + logtext + 'YES')        # for Debug
459    else:    # HAM
460        logtext = " HAM
          ↪  ;"+str(y_hat[0][0])+";"+msglength+";"+version+";"+lang+
          ↪  ";"+str(langP)+";"+mod+";"+sender+";"+recipient+"; "
461        message.add_header('X-KISPAM: ' + logtext, 'NO')        # for
          ↪  Debug
462        if(verbose):
463            print('X-KISPAM: ' + logtext + 'NO')        # for Debug
464    logger.info(logtext)
465    p = subprocess.Popen(['/usr/sbin/sendmail', '-oi', '-f', sender,
        ↪  recipient], stdin=subprocess.PIPE, stderr=subprocess.PIPE)
466    stdout = p.communicate(input=message.as_bytes())
```

Listing 3: CheckAntispam.py Das ist der eigenliche SPAM-Klassifier.

A.1.2. mail2rmailSkript.py

```
# mail2rmailSkript.py Soll ein Skript erzeugen, dass
#* schauen welche Mailbackup es im aktuellen Verz gibt dann, dann
    ein Skript erzeugen das:
#** eine Mail-Backup entpackt
#** das Skript rmail drueberlauffen laesst damit das CSV erzeugt
    wird
#** dann das entpackte Verz loeschen
#* Am Ende alle CSVs zu einem neuen zusammenfassen
import os
from pathlib import Path     # for reding dirs
from shutil import copyfile # for copy
import sys                   # cmd input parameter

if(len(sys.argv) == 2):
    try:
        arg = sys.argv[1:]
        for a in arg:
            if(a == "-h"):
                print(" only HAM")
                ham     = True
            elif(a == "-s"):
                print(" only SPAM")
                ham = False
            elif(a == "-v"):
                print("Verbose Output")
                verbose = True
            else:                # error
                print("-s fuer SPAM oder ODER -h fuer HAM ist
                    noetig!")
                sys.exit()
    except IndexError:
        raise SystemExit(f"Usage: {sys.argv[0]} <-v> <-[h|s]>
            <path>")
        print(arg[::-1])
```

```
31  else:
32      print("-s fuer SPAM oder ODER -h fuer HAM ist noetig!")
33      sys.exit()
34  
35  onlyOnce = False
36  outputName = "noName."
37  dirPath = "."
38  skriptText = []
39  skriptText.append("#!/bin/bash")
40  
41  fileList= Path(dirPath).glob("*.mail.tgz")
42  for fileName in fileList:
43      decText = "tar -xvzf " + str(fileName)
44      skriptText.append(decText)
45      pathMailArchiv = str(fileName).split('m')[0]
46      if(onlyOnce == True):
47          outputName = pathMailArchiv
48      pathMailArchiv = pathMailArchiv[0:-1] + '/'      # remove
      ↪ point add Slash
49      if(ham == True):
50          skriptText.append("python3 rmailbox_maildir_read.py " +
          ↪ pathMailArchiv + " -h")
51      else:   # spam
52          skriptText.append("python3 rmailbox_maildir_read.py " +
          ↪ pathMailArchiv+ " -s")
53      rmText = "rm -R " + pathMailArchiv
54      skriptText.append(rmText )
55  skriptText.append("cat *.csv > " + outputName + "csv2")
56  
57  for line in skriptText:
58      print(line)
```

Listing 4: mail2rmailSkript.py Dieses Skript erstellt ein Bash-Skript für die Umwandlung der Mailarchivdatein in CSV.

A.1.3. rmailbox_csv_read.py

```
1  # python3
2  # rmailbox_csv_read.py
3  #    Wandelt selbst erstellte CSVs ins richtige Format um
4  import mailbox
5  import quopri    # for decode
6  from email.header import decode_header
7  from bs4 import BeautifulSoup       # html2text pip3 install
   ↪  BeautifulSoup4
8  import csv           # For output
9  import sys           # cmd input parameter
10 import difflib       # compare plain 2 html
11 import re            # for removing URLs
12 import pandas as pd # for Working with Pandas
13
14 maxMail = 10000
15 ham      = False
16 spam     = False
17 verbose = False
18 mboxPath = '2021-09-23-14-58-49.2021-09-24-00-03-48'
19 if(len(sys.argv) > 1):
20     try:
21         arg = sys.argv[1:]
22         for a in arg:
23             if(a == "-h"):
24                 print("only HAM")
25                 ham      = True
26             elif(a == "-s"):
27                 print("only SPAM")
28                 spam     = True
29             elif(a == "-v"):
30                 print("Verbose Output")
31                 verbose = True
32             else:                    # is Path
33                 mboxPath = a
34                 print("a: ",a)
35     except IndexError:
36         raise SystemExit(f"Usage: {sys.argv[0]} <-v> <-[h|s]>
           ↪  <path>")
```

```
37      print(arg[::-1])
38  if(mboxPath[-1] == "/" ):
39      mboxPath = mboxPath.replace('/','')
40      if(verbose):
41          print("neuer Pfad: ",mboxPath)
42  mbox = pd.read_csv(mboxPath, sep =',',names=["label", "message"],
    ↪   header = 0)
43  zm = mbox.__len__()    # so bekommt man auch die Anzahl der
    ↪   Nachrichten in den Ordner oder Mailbox raus
44  z = 0
45  za = 0       # count ausgehende Mails
46  zu = 0       # unicode error
47  zs = 0       # Anzahl mit Subject
48  ze = 0       # Anzahl eingehende Mails
49  zSPAM = 0     # SPAM Anzahl
50  csvList = []
51  for message in mbox["message"] :
52      if(verbose):
53          print(type(message ))
54      text = message
55      if(isinstance(message,str)):
56          z = z + 1
57          if(verbose):
58              print(text[0:10])
59          if(verbose):
60              print("-----------------------------------------------
                ↪  --     -     -    ")
61          if(verbose):
62                  print(len(text))
63          if(len(text) > 32000):
64                          text = text[0:31999]
65                          if(verbose):
66                              print(" tolong->Cut")
67          text= text.replace('\\n',' ')   # newline und carrage
            ↪  retrun entfernen
68          text= text.replace('\n',' ')
69          text= text.replace('\\r',' ')
70          text= text.replace('\r',' ')
71          text= text.replace('\\t',' ')
```

```
72          text= text.replace('\t',' ')
73          text = re.sub('http[s]?://(?:[a-zA-Z]|[0-9]|[$-
              ↪  _@.&+]|[!*\(\),]|(?:%[0-9a-fA-F][0-9a-fA-F]))+', '',
              ↪  text, flags=re.MULTILINE)
74          text=text.replace(';',' ')
75          text=text.replace('"',' ')
76          text=text.replace('`',' ')
77          text=text.replace("'",' ')
78          text=text.replace(':',' ')
79          text= text.replace('      ',' ')
80          text= text.replace('    ',' ')
81          text= text.replace('   ',' ')
82          text= text.replace('  ',' ')
83          text= text.replace('_',' ')
84          text= text.replace('-',' ')
85          text= text.replace('|',' ')
86          if(len(text) > 200):
87                          if(verbose):
88                              print(" ", text[0:200])
89          else:
90                          if(verbose):
91                              print(" ", text)
92          if(ham):
93              hamSpam = "ham"
94          else:
95              hamSpam = "spam"
96          if(verbose):
97              print(len(text))
98          if(len(text) > 1):
99              if(len(text) > 32000):
100                 csvList.append([hamSpam,text[0:31999]])
101             else:
102                 csvList.append([hamSpam,text])
103         if(z >= maxMail):       # sieht nur die ersten x Mails an
104             break
105         if(verbose):
106             print(len(csvList)," Anzahl csvList")
107     else:
108         print(type(message))
```

```
109 if(verbose):
110     print("- -- --- --- -- -- -- -Ausgabe- - -- - -- -- - -")
111     if(verbose):
112         print(len(csvList)," Anzahl csvList")
113 with open(mboxPath+'.csv', mode='w', newline='') as csv_file:
114     csv_writer = csv.writer(csv_file, delimiter=',')
115     za = 0
116     for mail in csvList:
117         za = za + 1
118         csv_writer.writerow(mail)
119 print("\nMailanzahl:\t", zm, "\nbearbeitet:\t",
     ↪  z,"\nausgegeben:\t",za )
```

Listing 5: rmailboxCSVread.py Dieses Skript wird vom zuvor erstellten Bash-Skript für die Umwandlung verendet.

A.2. Längen- und Spracherkennung

A.2.1. labelCSVmail.ipynb

```
1  import numpy as np
2  import pandas as pd
3  import seaborn as sns
4  import matplotlib.pyplot as plt
5  from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
6  %matplotlib inline
7
8  from sklearn.model_selection import train_test_split # library
    ↪  for train test split
9  from sklearn.metrics import roc_auc_score # for roc_auc_score
10
11 import os # Reading and writing from Disk
12 import fasttext  # # Language Detection
13
```

```
14 filename = "ham2021spam4Jahre.csv"
15 messages = pd.read_csv(filename, sep =',',names=["label",
   ↪ "message"], header = 0)
16 PRETRAINED_MODEL_PATH = 'lid.176.bin'  # fastTextIstDochSpam.csv
17 modelLang = fasttext.load_model(PRETRAINED_MODEL_PATH)
18 print(messages.head())
19
20 type(messages)
21 messages.info()
22 messages.shape
23 messages.size
24 messages.dtypes
25 messages.columns
26
27 messages=messages.drop_duplicates()        # Delete duplicates
28 messages.describe()
29
30 print(messages.head())        # How is the length of the messages
   ↪ in char distributed
31 avglength = sum(messages.message.str.len())/messages.size
32 allength = messages.message.str.len()
33 allength.describe(percentiles=[.1,.2,.3,.4,.5,.6,.7,.8,.9])
34
35 langList = []
36 lenList = []
37 for msg in messages['message']:        # save Message Language
   ↪ and Length in 3 categories
38         lang = modelLang.predict(str(msg))[0][0]
39         print(lang)
40         langList.append(lang)
41         msg = str(msg)
42         if(len(msg) <= 1500):
43             print("short")
44             lenList.append("short")
45         elif(len(msg) < 3000):
46             print("middle")
47             lenList.append("middle")
48         else:
49             print("long")
```

```
50              lenList.append("long")
51
52  messages["lang"] = langList
53  messages["len"] = lenList
54  print(messages)
55
56  messages.describe().T
57
58  messages.groupby('label').describe()
59
60  messages.groupby('lang').describe()
61
62  messages['lang'].value_counts().head(10)
63
64  messages.groupby('len').describe()
65
66  outputFilename = "lab_" + filename
67  messages.to_csv(outputFilename)
```

Listing 6: labelCSVmail.ipynb Hier werden der Maildatensatz mit den Zusatzinformationen, Sprache und Maillänge angereichert.

A.3. Nilsimsa

Der Nilsimsa-Hash wird berechnet und als zusätzliche Spalte im Mail-Datensatz gespeichert.

A.3.1. nilLab3.ipynb

```
1  # * nilLab3 soll
2  # * * die Mails im lab3 Format einlesen
3  # * * nil berechnen
4  # * * und im labt Format speichern (ohne original
   ↪   spam/ham/Translated/lang/wahrscheinlichkeit/laenge)
5  # * * Ausgabe in labt_Originalname.csv
```

```
6
7  import numpy as np
8  import pandas as pd
9  from nilsimsa import Nilsimsa,compare_digests
10
11 import os # Reading and writing from Disk
12 import time # take the time
13 filename = "lab3_6jSpamHam1q24.csv"
14 messages = pd.read_csv(filename, sep =',', header = 0) #
   ↪ label,message,lang,prop,len,translated
15 print(messages.head())
16
17 type(messages)
18 messages.info()
19 messages.shape
20 messages.size
21 messages.dtypes
22 messages.columns
23
24 messages=messages.drop_duplicates().reset_index(drop=True)
25 messages.shape
26
27 list(messages.columns.values)
28
29 def getHash(data:str)-> str:
30     nil=Nilsimsa(str(data).encode())
31     return nil.hexdigest()
32
33 def compPrint(sample1 ,sample2):
34     hash1 = getHash(sample1)
35     hash2 = getHash(sample2)
36     print(compare_digests(hash1,hash2))
37
38 def compPrintN(sample1 ,sample2, sample1N, sample2N):
39     hash1 = getHash(sample1)
40     hash2 = getHash(sample2)
41     result = (compare_digests(hash1,hash2))
42     print(str(result) + "\t" +sample1N+ "/\t" + sample2N)
43     return result
```

```
44
45 def compPrintL(listText):
46     deleted = 0
47     for i in range(len(listText)):
48         for j in range(i + 1, len(listText)):
49             res = compPrintN(listText[i],
                ↪  listText[j],str(i+deleted),str(j+deleted))
50             if(res==128): # we have a match delete this entry, do
                ↪  not move with the comparing
51                 print("deleted: "+str(i+deleted))
52                 listText.pop(i)
53                 deleted = deleted + 1
54                 break
55     return(listText)
56
57 nilList = []
58 start_time = time.time()
59 for index,msg in messages.iterrows():
60     nilHash = getHash(msg['message'])
61     nilList.append(nilHash)
62     if (index % 100 +1  == 1):
63         print(str(index+1)+"\t"+nilHash) # es gibt welche die
            ↪  sind leer!
64 end_time = time.time()
65 training_timeS = (end_time  - start_time)
66 training_time = (training_timeS) / 60
67 hashPerS = (index+1)/training_timeS
68 print(f'Total algo time: {training_time:.2f} min {hashPerS:.2f}
    ↪  hash/s')
69
70 len(nilList)
71
72 messages["nil"] = nilList # fuegt Spalte hinzu
73 messagesOut = messages
74 outputFilename = "labt_" + filename
75 messagesOut.to_csv(outputFilename)
```

Listing 7: nilLab3.ipynb

A.3.2. nilLabChoose.ipynb

```
# ### * Soll
# * * die Mails im lab Format einlesen
# * * die aehnlichen Mails löschen
# * * und im labt Format speichern (ohne
↪   original,spam/ham,Translated,lang,wahrscheinlichkeit,laenge)
# * * Ausgabe in labt_Originalname.csv

import numpy as np
import pandas as pd
from nilsimsa import Nilsimsa,compare_digests
import os # Reading and writing from Disk
import time # take the time

filename = "lab3_1monU65k.csv"
nilBorder = 31 # # 0-127, bei 12345 wurde mit 100 rund 1/3
↪   geloescht, 63 bleiben rund 1/4 übrig
messages = pd.read_csv(filename, sep =',', header = 0) #
↪   label,message,lang,prop,len,translated
print(messages.head())

type(messages)
messages.info()
messages.shape
messages.size
messages.dtypes
messages.columns

messages.describe()

messages=messages.drop_duplicates().reset_index(drop=True)
messages.describe()

list(messages.columns.values)

```

```
32  messages.groupby('label').describe()
33
34  messages.groupby('lang').describe()
35
36  langList=messages.lang.unique()
37  langList
38
39  lenList=messages.len.unique()
40  lenList
41
42  messages.groupby('prop').describe()
43
44  propPerc=messages.prop.describe(percentiles=[.05,.1,.15,.2,.25,.3,.35,.4,.45
45  propPerc
46
47  messages.groupby('len').describe()
48
49  def getHash(data:str)-> str:
50      nil=Nilsimsa(data.encode())
51      return nil.hexdigest()
52
53  def compPrint(sample1 ,sample2):
54      hash1 = getHash(sample1)
55      hash2 = getHash(sample2)
56      print(compare_digests(hash1,hash2))
57
58  def compPrintN(sample1 ,sample2, sample1N, sample2N):
59      hash1 = getHash(sample1)
60      hash2 = getHash(sample2)
61      result = (compare_digests(hash1,hash2))
62      print(str(result) + "\t" +sample1N+ "/\t" + sample2N)
63      return result
64
65  def compPrintL(listText):
66      deleted = 0
67      for i in range(len(listText)):
68          for j in range(i + 1, len(listText)):
69              res = compPrintN(listText[i],
                 ↪  listText[j],str(i+deleted),str(j+deleted))
```

```
70                  if(res==128): # we have a match delete this entry, do
                     ↪  not move on with the comparing
71                      print("deleted: "+str(i+deleted))
72                      listText.pop(i)
73                      deleted = deleted + 1
74                      break
75      return(listText)
76
77  nilList = []
78  start_time = time.time()
79  for index,msg in messages.iterrows():
80      nilHash = getHash(msg['message'])
81      nilList.append(nilHash)
82      print(str(index+1)+"\t"+nilHash) # es gibt welche die sind
         ↪  leer!
83  end_time = time.time()
84  training_timeS = (end_time  - start_time)
85  training_time = (training_timeS) / 60
86  hashPerS = (index+1)/training_timeS
87  print(f'Total algo time: {training_time:.2f} min {hashPerS:.2f}
     ↪  hash/s')
88
89  len(nilList)
90
91  messages["nil"] = nilList # fuegt Spalte hinzu
92
93  def compDropL(listText): # erkennt aehndliche Mails
94      deleted = 0
95      dropList = []
96      for i in range(len(listText)):
97          for j in range(i + 1, len(listText)):
98              res = compare_digests(listText[i], listText[j])
99              if(res > nilBorder): # we have a match delete this
                     ↪  entry, do not move on with the comparing
100                     dropList.append(i)
101                     deleted = deleted + 1
102                     break
103     return(dropList)
104
```

```
105 shortList = []
106 for x in range(len(lenList)):
107     for y in range(len(langList)):
108         for z in range(3, len(propPerc)-1): # start at 3 end at
        ↪ 22, because of +1 for max value
109             shortD =
            ↪ messages.loc[(messages.len==lenList[x])&(messages.lang==lang
110             shortDF = shortD.copy()
111             shortDF.reset_index(drop=True)
112             shortList.append(shortDF)
113 
114 start_time = time.time()
115 dropped = 0
116 for x in range(len(shortList)):
117     dropList = compDropL((shortList[x])['nil'].tolist())
118     end_time  = time.time()
119 
    ↪ shortList[x].drop(shortList[x].index[dropList],inplace=True)#.reset_
120     training_time = (end_time  - start_time)/60
121     dropped = dropped + len(dropList)
122 print(f'Total algo time: {training_time:.2f} min, {dropped}
 ↪ dropped') # time for calc
123 
124 messagesOut = pd.concat(shortList)
125 messagesOut
126 
127 messagesOut.describe()
128 
129 outputFilename = "labt_" + filename
130 messagesOut.to_csv(outputFilename)
```

Listing 8: nilLabChoose.ipynb Mails werden je Nilsimsa-Grenzwert aussortiert, der Mail-Datensatz wird damit hauptsächlich um ähnliche Mails kleiner.

A.4. Verhältnis anpassen

A.4.1. nilLabDiffnil-2x.ipynb

```
# * Soll (damit spater nur mehr die neuen mails ubersetzt werden)
# * * 2x die Mails im lab Format einlesen
# * * dann den Unterschied in einer Datei speichern
# * * Ausgabe in diff_Originalname_OriginalName2.csv
import numpy as np
import pandas as pd
from nilsimsa import Nilsimsa,compare_digests

import os # Reading and writing from Disk
import time # take the time

get_ipython().system(' pip list')

filename = "labt_v127_lab3_6jSpamHam1q24_nil121.csv"
filename2 = "labt_v122_lab3_6jSpamHam1q24_nil121.csv"

messages = pd.read_csv(filename, sep =',', header = 0) #
  ↪ 'Unnamed: 0', 'Unnamed: 0.1', 'label', 'message', 'lang',
  ↪ 'prop', 'len',       'nil'
messages2 = pd.read_csv(filename2, sep =',', header = 0) #
  ↪ Unnamed: 0', 'nr', 'Unnamed', 'label', 'message', 'lang',
  ↪ 'prop',       'len', 'nil', 'translated'

type(messages)
messages.info()
messages.shape
messages.size
messages.dtypes

type(messages2)
messages2.info()
messages2.shape
messages2.size
messages2.dtypes

```

```
32  list(messages.columns.values)
33
34  list(messages2.columns.values)
35
36  def getHash(data:str)-> str:
37      nil=Nilsimsa(data.encode())
38      return nil.hexdigest()
39
40  def compPrint(sample1 ,sample2):
41      hash1 = getHash(sample1)
42      hash2 = getHash(sample2)
43      print(compare_digests(hash1,hash2))
44
45  def compPrintN(sample1 ,sample2, sample1N, sample2N):
46      hash1 = getHash(sample1)
47      hash2 = getHash(sample2)
48      result = (compare_digests(hash1,hash2))
49      print(str(result) + "\t" +sample1N+ "/\t" + sample2N)
50      return result
51
52  def compPrintL(listText):
53      deleted = 0
54      for i in range(len(listText)):
55          #print (str(i)+ ": "+ listText[i])
56          for j in range(i + 1, len(listText)):
57              res = compPrintN(listText[i],
                    ↪  listText[j],str(i+deleted),str(j+deleted))
58              if(res==128): # we have a match delete this entry, do
                    ↪  not move on with the comparing
59                  print("deleted: "+str(i+deleted))
60                  listText.pop(i)
61                  deleted = deleted + 1
62                  break
63      return(listText)
64
65  deleted = 0
66  dropList = []
67  start_time = time.time()
68  for index2,msg2 in messages2.iterrows():
```

```
69        for index,msg in messages.iterrows():
70            if(messages["nil"] == messages2["nil"] ):
71                    dropList.append(msg2["Unnamed: 0.1"]) # compare
                       ↪ just this number, they are the same
72                    deleted = deleted + 1
73
                       ↪ #print(str(i)+"/"+str(j)+"d"+str(deleted)+"l"+str(len(list
74    end_time = time.time()
75    training_timeS = (end_time  - start_time)
76    training_time = (training_timeS) / 60
77    hashPerS = (index2+1)/training_timeS
78    print(f'Total algo time: {training_time:.2f} min {hashPerS:.2f}
       ↪ hash/s')
79    len(dropList)
80
81    dropList[0:9]
82
83    messages.drop("Unnamed: 0", axis="columns" ,inplace=True)
84    messages.set_index("Unnamed: 0.1",inplace=True)
85    messages.drop(dropList ,inplace=True , errors="ignore")
86    messages.head(5)
87    messages.shape
88
89    outputFilename = "diff_" + filename + "_" + filename2
90    messages.to_csv(outputFilename)
```

Listing 9: nilLabDiffnil-2x.ipynb Vergleich 2 Mail-Datensätze und entfernt ähnliche Mails.

A.5. Stemmer, Lemma, LPD, und Übersetzen

A.5.1. replaceNER.ipynb

```
# * Soll NER Bearbeiteung machen (ist bei labelCSV Translate
↪ schon eingebaut)
# * * die Mails im CSV-Format einlesen
# * * aus der Spalte translated bearbeiten (Space-Model ist je
↪ Sprache, daher nur mit den UEbersetzten arbeiten)
# * * Mailadressen URLs und Zahlen durch Allgmeines ersetzten
# * * Ausgabe in NER_Originalname.csv

import numpy as np
import pandas as pd

import os # Reading and writing from Disk
import spacy # auf saetzte aufteilen

import fasttext  # # Language Detection
import argostranslate.package # Translation
import argostranslate.translate # Translation
import time # take the time
import re # regex
from itertools import groupby # fuer: baut leerzeichen bei zahlen
↪ ein, fals worter dran haengen

filename = "6jSpamHam1q24_v46.csv"

# messages = pd.read_csv(filename, sep =',',names=["label",
↪ "message"], header = 0) # csv ohne lab
messages = pd.read_csv(filename, sep =',', header = 0)  # lab
PRETRAINED_MODEL_PATH = 'lid.176.bin'  # fastTextIstDochSpam.csv
modelLang = fasttext.load_model(PRETRAINED_MODEL_PATH)

type(messages)
messages.info()
messages.size
messages.dtypes
messages.columns
messages.shape

messages=messages.drop_duplicates().reset_index(drop=True)
messages.shape
```

```
36
37 source_lang = spacy.load("xx_sent_ud_sm") # ['senter']
38
39 source_lang.pipe_names # list pipes
40
41 # html cleaner, wurde eigentlich schon mit BeautifullSoap gemacht
   ↪ aber es sind Fragmente übrig geblieben, die schluckt die
   ↪ Übersetzung nicht
42 CLEANR = re.compile('<.*?>')
43 def cleanhtml(raw_html):
44   cleantext = re.sub(CLEANR, '', raw_html)
45   return cleantext
46
47 get_ipython().system('date')
48
49 NERList = []
50 start_time = time.time()
51 for index,msg in messages.iterrows():
52         one = msg['NER']  # if NER schon tw. gemacht
53         one = cleanhtml(str(one)) # remove html tags
54         one = ''.join([' ' + i + ' ' if i.isdigit() else i for k,
            ↪ g in groupby(one) for i in g]) # leerzeichen bei
            ↪ zahlen
55         doc = source_lang(str(one)) # Process the source text
56         wOutput = 0 # words in the output file, early stop if
            ↪ more then 550 reached
57         lenSent = 0
58         wordsStr = " "
59         for sent in doc.sents: # Iterate over sentences in the
            ↪ processed text
60             words = []
61             for token in sent:
62                 word = ''
63                 if (token.is_space == True):
64                     pass
65                 elif (token.is_punct == True):
66                     pass
67                 elif (token.is_digit == True):
68                     word = '1'
```

```
                    elif (token.like_num == True):
                        word = '1'
                    elif (token.like_email == True):
                        word = 'a@b.c'
                    elif (token.like_url == True):
                        word ='https://a.b/c'
                    else:
                        word = str(token)
                    words.append(word)
                wordsStr = wordsStr + " ".join(words) + " "
            NERList.append(wordsStr)
end_time = time.time()
training_time = (end_time - start_time) / 60
print(f'Total translation time: {training_time:.2f} min')

get_ipython().system('date')

messages["NER"] = NERList

outputFilename = "NERrSingle_" + filename
messages.to_csv(outputFilename)
print(outputFilename)
```

Listing 10: replaceNER.ipynb Ersetzt Mailadressen, Urls und Zahlen im Mail-Datensatz durch Platzhalter.

A.5.2. removeNER.ipynb

```
# * Soll NER Bearbeiteung machen (ist bei labelCSV Translate
    schon eingebaut)
# * * die Mails im CSV-Format einlesen
# * * aus der Spalte translated bearbeiten (Space-Model ist je
    Sprache, daher nur mit den UEbersetzten arbeiten)
# * * Mailadressen URLs und Zahlen loeschen
# * * Ausgabe in NER_Originalname.csv
```

```
6
7  import numpy as np
8  import pandas as pd
9
10 import os # Reading and writing from Disk
11 import spacy # auf saetzte aufteilen
12
13 import fasttext  # # Language Detection
14 import argostranslate.package # Translation
15 import argostranslate.translate # Translation
16 import time # take the time
17 import re # regex
18 from itertools import groupby # fuer: baut leerzeichen bei zahlen
   ↪  ein, fals worter dran haengen
19
20 filename = "6jSpamHam1q24_v46.csv"
21 messages = pd.read_csv(filename, sep =',', header = 0)  # lab
22 PRETRAINED_MODEL_PATH = 'lid.176.bin'  # fastTextIstDochSpam.csv
23 modelLang = fasttext.load_model(PRETRAINED_MODEL_PATH)
24
25 type(messages)
26 messages.info()
27 messages.size
28 messages.dtypes
29 messages.columns
30 messages.shape
31
32 messages=messages.drop_duplicates().reset_index(drop=True)
33 messages.shape
34
35 source_lang = spacy.load("xx_sent_ud_sm") # ['senter']
36 source_lang.pipe_names # list pipes
37
38 # html cleaner, wurde eigentlich schon mit BeautifullSoap gemacht
   ↪  aber es sind Fragmente übrig geblieben, die schluckt die
   ↪  Übersetzung nicht
39 CLEANR = re.compile('<.*?>')
40 def cleanhtml(raw_html):
41   cleantext = re.sub(CLEANR, '', raw_html)
```

```
42     return cleantext
43
44   NERList = []
45   start_time = time.time()
46   for index,msg in messages.iterrows():
47           one = msg['NER']  # if NER schon tw. gemacht
48           one = cleanhtml(str(one)) # remove html tags
49           one = ''.join([' ' + i + ' ' if i.isdigit() else i for k,
           ↪  g in groupby(one) for i in g]) # leerzeichen bei
           ↪  zahlen
50           one = one.replace("mailto"," ")
51           one = one.replace("imageurl"," ")
52           one = one.replace("isbanner"," ")
53           doc = source_lang(str(one)) # Process the source text
54           wOutput = 0 # words in the output file, early stop if
           ↪  more then 550 reached
55           lenSent = 0
56           wordsStr = " "
57           for sent in doc.sents: # Iterate over sentences in the
           ↪  processed text
58               words = [token.text for token in sent if
               ↪  token.is_stop != True and token.is_punct != True
               ↪  and token.is_space != True  and token.is_digit !=
               ↪  True and token.like_url != True  and
               ↪  token.like_url != True  and token.like_num !=
               ↪  True  and token.like_email != True ]
59               wordsStr = wordsStr + " ".join(words) + " "
60           NERList.append(wordsStr)
61   end_time = time.time()
62   training_time = (end_time - start_time) / 60
63   print(f'Total translation time: {training_time:.2f} min')
64
65   messages["NER"] = NERList
66   print(messages.head())
67
68   outputFilename = "NER53_" + filename
69   messages.to_csv(outputFilename)
```

Listing 11: removeNER.ipynb Löscht Mailadressen, Urls und Zahlen aus dem Mail-Datensatz .

A.5.3. antispamSpacy.ipynb

```
1  # * 11 09102024 spacy trf Modell fuer DE und EN
2  #     Traning mit lemma danach durchgefuehrt
3  import tensorflow as tf # braucht CPU mit avx flag
4  print(tf.__version__)
5
6  # import libarries for reading data, exploring and plotting
7  import pickle   # Load Save Tokenizer
8  import numpy as np
9  import pandas as pd
10 import seaborn as sns
11 import matplotlib.pyplot as plt
12 from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
13 get_ipython().run_line_magic('matplotlib', 'inline')
14
15 from sklearn.model_selection import train_test_split # library
    ↪ for train test split
16 from sklearn.metrics import roc_auc_score # for roc_auc_score
17
18 import tensorflow as tf # deep learning libraries for text
    ↪ pre-processing
19 from tensorflow.keras.preprocessing.text import Tokenizer
20 from tensorflow.keras.preprocessing.sequence import pad_sequences
21
22 # Modeling
23 from tensorflow.keras.callbacks import EarlyStopping
24 from tensorflow.keras.models import Sequential
25 from tensorflow.keras.layers import Embedding,
    ↪ GlobalAveragePooling1D,  Flatten, Dense, Dropout, Embedding,
    ↪ LSTM, SpatialDropout1D, Bidirectional, SpatialDropout1D,
    ↪ Conv1D, GlobalMaxPooling1D
26 from keras.callbacks import ModelCheckpoint
27 from keras.layers import Input, concatenate # Multi-ConvNet
28 from keras.models import Model
29 import os # Reading and writing from Disk
```

```
30  from tensorflow.keras.utils import plot_model # For Graphic of
    ↪  the DNN
31
32  msglength = "msg"   ### every
33  #msglength = "short"
34  #msglength = "middle"
35  #msglength = "long"
36  version = "44"
37  lang = "__label__de"
38  #lang = "__label__en"
39  #lang = "__label__somtehingElse"
40  #lang = "all"
41
42
43  filename = "lab2_mail65plus.csv"
44  messages = pd.read_csv(filename, sep =',',names=["label",
    ↪  "message","lang","prop","len","lang2","prop2"], header = 0)
45  MAX_LEN = 300 # pad_sequencs parameter, only look for X words in
    ↪  a sentence (around 80% of the sentences)
46
47  type(messages)
48  messages.info()
49  messages.shape
50  messages.size
51  messages.dtypes
52  messages.columns
53
54  messages=messages.drop_duplicates()
55  messages.shape
56
57
58  # How is the length of the messages in char distributed
59  avglength = sum(messages.message.str.len())/messages.size
60  allength = messages.message.str.len()
61  allength.describe(percentiles=[.1,.2,.3,.4,.5,.6,.7,.8,.9])
62
63
64  dropList = []
65  cnt = 0
```

```
66  if(msglength != "msg"):
67      for msg in messages['len']:
68          #print(msg)
69          if(msg != msglength):
70              dropList.append(cnt)
71          cnt = cnt + 1
72
73  messages.drop(messages.index[dropList], inplace=True) # deletes
     ↪ not used messages
74  messages.shape
75
76  dropList = []
77  cnt = 0
78  if(lang != "all"): # Alle Sprachen zu machen
79      if(lang == "__label__de" or lang == "__label__en"): # lang is
         ↪ something else
80          for msg in messages['lang']:
81              #print(msg)
82              if(msg != lang):
83                      dropList.append(cnt)
84              cnt = cnt + 1
85      else:   # Language is de or en
86          for msg in messages['lang']:
87              #print(msg)
88              if(msg == "__label__de" or msg == "__label__en"):
89                      dropList.append(cnt)
90              cnt = cnt + 1
91  messages.drop(messages.index[dropList], inplace=True) # deletes
     ↪ not used messages
92  messages.shape
93
94  messages.groupby('label').describe().T
95
96  ham_msg = messages[messages.label == 'ham'] # get all the ham and
     ↪ spam emails
97  spam_msg = messages[messages.label == 'spam']
98  ham_msg_text = " ".join(ham_msg.message.to_numpy().tolist()) #
     ↪ For ham and spam messages, create numpy list
99  spam_msg_text = " ".join(spam_msg.message.to_numpy().tolist())
```

```
100
101 plt.figure(figsize=(8,6))
102 sns.countplot(messages.label)
103 plt.title('Distribution of ham and spam email messages')
104
105 # Percentage of spam messages
106 (len(spam_msg)/len(ham_msg))*100
107
108 # Dowsample the HAM so that is 1:1 with HAM. (wenns lange dauert,
    ↪ war MAX_FAIl zu grosz)
109 if(len(ham_msg) > len(spam_msg)):  # one way to fix it is to
    ↪ downsample the ham msg
110     ham_msg_df = ham_msg.sample(n = len(spam_msg), random_state =
        ↪ 44)
111     spam_msg_df = spam_msg
112 else:# more spam then ham
113     spam_msg_df = spam_msg.sample(n = len(ham_msg), random_state
        ↪ = 44)
114     ham_msg_df = ham_msg
115 # Create a dataframe with these ham and spam msg
116 msg_df = ham_msg_df.append(spam_msg_df).reset_index(drop=True)
117
118 plt.figure(figsize=(8,6))
119 sns.countplot(msg_df.label)
120 plt.title('Distribution of ham and spam email messages (after
    ↪ downsampling)')
121 plt.xlabel('Message types')
122
123 # Get length column for each text
124 msg_df['text_length'] = msg_df['message'].apply(len)
125 #Calculate average length by label types
126 labels = msg_df.groupby('label').mean()
127 labels
128
129
130 # ## Prepare data
131 # Convert text labels to numeric. Save 0 for Ham and 1 for SPAM
132 msg_df['msg_type']= msg_df['label'].map({'ham': 0, 'spam': 1})
133 msg_label = msg_df['msg_type'].values
```

```
134
135 # How many Words are in one Message?
136 wordsInMessages = []
137 for msg in msg_df['message']:        # Words in Message
138     wordsInMessages.append(len(msg.split()))
139 np_wordsInMessages = np.array(wordsInMessages)        # to numpy
     ↪ array
140 df_wordsInMessages = pd.DataFrame(np_wordsInMessages, columns =
     ↪ ['CntWords']) # to DataFrame
141 df_wordsInMessages.describe(percentiles=[.1,.2,.3,.4,.5,.6,.7,.8,.9])
142
143 import spacy # RUN spacy download "en_core_web_sm" # Lemma sollte
     ↪ besser sein als Stemmer
144 #nlpE = spacy.load("en_core_web_sm") # small
145 #nlpDE = spacy.load("de_core_news_sm") # small
146 nlpE = spacy.load("en_core_web_trf") # transformer
147 nlpDE = spacy.load("de_dep_news_trf") # transformer
148 nlpSO = spacy.load("xx_ent_wiki_sm") # small
149
150
151 # Stemmer
152 #from nltk.stem import SnowballStemmer
153 #snowball = SnowballStemmer(language='english')
154 #
155 #words = ["running", "runner", "runs", "swimming", "swimmer",
     ↪ "swims"]
156 #stemmed_words = [snowball.stem(word) for word in words]
157 #
158 #print("Snowball Stemmer Output:")
159 #for original, stemmed in zip(words, stemmed_words):
160 #    print(f"{original} -> {stemmed}")
161
162 msg_dfSpacy = []
163
164 for row in msg_df.itertuples(index=True, name='Pandas'):
165     msg = row[2]
166     msgTxt = msg.lower() # only lower text
167     word_spacy = " "
168     doc = ""
```

```
169         if("__label__de" == row[3]):
170             doc = nlpDE(msgTxt)
171         elif("__label__en" == row[3]):
172             doc = nlpE(msgTxt)
173         else:
174             doc = nlpSO(msgTxt)
175         word_spacy = " ".join([token.lemma_ for token in doc if not
          ↪   token.is_punct]) # remove punction
176         msg_dfSpacy.append(word_spacy)
177 
178 print((msg_dfSpacy)[1:3])
179 msg_df["mesSpacy"] = msg_dfSpacy
180 
181 train_msg, test_msg, train_labels, test_labels =
      ↪   train_test_split(msg_df['mesSpacy'], msg_label,
      ↪   test_size=0.2, random_state=434)
182 
183 # pre-processing hyperparameters
184 trunc_type = "post" # pad_sequences parameter
185 padding_type = "post" # pad_sequences parameter
186 oov_tok = "<OOV>" # out of vocabulary token
187 VOCAB_SIZE = 15626   # need a fix size
188 
189 tokenizer = Tokenizer(num_words = VOCAB_SIZE, char_level=False,
      ↪   oov_token = oov_tok)
190 tokenizer.fit_on_texts(train_msg)
191 
192 # Get the word_index
193 word_index = tokenizer.word_index
194 word_index
195 
196 # check how many words
197 tot_words = len(word_index)
198 #VOCAB_SIZE = int(tot_words/4)    # Voacabulary Size 25%,    10%
      ↪   from all uniq Words by SMS
199 print('There are %s unique tokens in training data. ' %
      ↪   tot_words)
200 
201 tokenizer = Tokenizer(num_words = VOCAB_SIZE, char_level=False,
      ↪   oov_token = oov_tok)
```

```
202 tokenizer.fit_on_texts(train_msg)
203 output_dir = "model_output/" + msglength + version + lang +"/"
204 if not os.path.exists(output_dir):
205     os.makedirs(output_dir)
206 with open(output_dir + 'tokenizer.pickle', 'wb') as handle:
207     pickle.dump(tokenizer, handle,
     ↪   protocol=pickle.HIGHEST_PROTOCOL)
208 
209 # Sequencing and padding on training and testing
210 training_sequences = tokenizer.texts_to_sequences(train_msg)
211 training_padded = pad_sequences (training_sequences, maxlen =
 ↪   MAX_LEN,
212                                     padding = padding_type,
                                     ↪   truncating = trunc_type )
213 
214 testing_sequences = tokenizer.texts_to_sequences(test_msg)
215 testing_padded = pad_sequences(testing_sequences, maxlen =
 ↪   MAX_LEN,
216                                 padding = padding_type, truncating
                                 ↪   = trunc_type)
217 
218 # Shape of train tensor
219 print('Shape of training tensor: ', training_padded.shape)
220 print('Shape of testing tensor: ', testing_padded.shape)
221 
222 print(training_sequences[0])
223 
224 len(training_sequences[0]), len(training_sequences[1])         #
 ↪   Before padding : first sequence has a different length of the
 ↪   second
225 
226 len(training_padded[0]), len(training_padded[1])         # Padded
 ↪   to same length of 50
227 
228 print(training_padded[0])
229 print(training_sequences[0])
230 
231 # Dense sentiment model architecture
232 EMBEDING_DIM = 32  # 16 bei SMS
```

```
233 DROP_VALUE = 0.2 # dropout
234 N_DENSE = 24    #
235 model = Sequential()        #Dense sentiment model architecture
236 model.add(Embedding(VOCAB_SIZE, EMBEDING_DIM,
    ↪  input_length=MAX_LEN))
237 model.add(GlobalAveragePooling1D())
238 model.add(Dense(N_DENSE, activation='relu'))
239 model.add(Dropout(DROP_VALUE))
240 model.add(Dense(1, activation='sigmoid'))
241 model.summary()
242 
243 plot_model(model, to_file="m.png", show_shapes=True,
    ↪  show_layer_names=True)
244 
245 from tensorflow import keras
246 optim = keras.optimizers.Adam(learning_rate=0.00001)
247 output_dir = "model_output/dense0" + msglength + version + lang
248 modelcheckpoint = ModelCheckpoint(filepath = output_dir
    ↪  +"/weights.{epoch:02d}.hdf5")
249 if not os.path.exists(output_dir):
250     os.makedirs(output_dir)
251 model.compile(loss='binary_crossentropy',optimizer='adam'
    ↪  ,metrics=['accuracy'])
252 num_epochs = 30 # fitting a dense spam detector model
253 early_stop = EarlyStopping(monitor='val_loss', patience=5)
254 history = model.fit(training_padded, train_labels,
    ↪  epochs=num_epochs, validation_data=(testing_padded,
    ↪  test_labels),callbacks = (modelcheckpoint, early_stop),
    ↪  verbose=1)
255 
256 metrics = pd.DataFrame(history.history)
257 metrics[:2]
258 metrics.rename(columns = {'loss': 'Training_Loss', 'accuracy':
    ↪  'Training_Accuracy','val_loss': 'Validation_Loss',
    ↪  'val_accuracy': 'Validation_Accuracy'}, inplace = True)
259 metrics[:2]
260 
261 def plot_graphs1(var1, var2, string):
262     metrics[[var1, var2]].plot()
```

```
263     plt.title('Dense Classifier: Training and Validation ' +
         ↪  string)
264     plt.xlabel ('Number of epochs')
265     plt.ylabel(string)
266     plt.legend([var1, var2])
267 plot_graphs1('Training_Loss', 'Validation_Loss', 'loss')
268 plot_graphs1('Training_Accuracy', 'Validation_Accuracy',
     ↪  'accuracy')
269 
270 N_LTSM = 20         #LSTM layer arcitecture hyperparameters
271 output_dir1 = "model_output/ltsm" + msglength + version + lang
272 modelcheckpoint = ModelCheckpoint(filepath = output_dir1 +
     ↪  "/weights.{epoch:02d}.hdf5")
273 if not os.path.exists(output_dir1):
274     os.makedirs(output_dir1)
275 model1 = Sequential()
276 model1.add(Embedding(VOCAB_SIZE, EMBEDING_DIM,
     ↪  input_length=MAX_LEN))
277 model1.add(LSTM(N_LTSM, dropout=DROP_VALUE,
     ↪  return_sequences=True))
278 model1.add(LSTM(N_LTSM, dropout=DROP_VALUE,
     ↪  return_sequences=True))
279 model1.add(GlobalAveragePooling1D())
280 model1.add(Dense(1, activation='sigmoid'))
281 model1.summary()
282 
283 plot_model(model1, to_file="m1.png", show_shapes=True,
     ↪  show_layer_names=True)
284 
285 model1.compile(loss = 'binary_crossentropy', optimizer = 'adam',
     ↪  metrics=['accuracy'])
286 
287 num_epochs = 30
288 early_stop = EarlyStopping(monitor='val_loss', patience=3)
289 history1 = model1.fit(training_padded, train_labels,
     ↪  epochs=num_epochs, validation_data=(testing_padded,
     ↪  test_labels), callbacks = (modelcheckpoint, early_stop),
     ↪  verbose=1)
290 
```

```
291 metrics = pd.DataFrame(history1.history)        # Create a
    ↪  dataframe
292 metrics.rename(columns = {'loss': 'Training_Loss', 'accuracy':
    ↪  'Training_Accuracy', 'val_loss': 'Validation_Loss',
    ↪  'val_accuracy': 'Validation_Accuracy'}, inplace = True)
293 def plot_graphs(var1, var2, string):
294     metrics[[var1, var2]].plot()
295     plt.title('LSTM Model: Training and Validation ' + string)
296     plt.xlabel ('Number of epochs')
297     plt.ylabel(string)
298     plt.legend([var1, var2])
299 plot_graphs('Training_Loss', 'Validation_Loss', 'loss')
300 plot_graphs('Training_Accuracy', 'Validation_Accuracy',
    ↪  'accuracy')
301
302 output_dir2 = "model_output/bi-ltsm" + msglength + version + lang
303 modelcheckpoint = ModelCheckpoint(filepath = output_dir2 +
    ↪  "/weights.{epoch:02d}.hdf5")
304 if not os.path.exists(output_dir2):
305     os.makedirs(output_dir2)
306
307 model2 = Sequential()        # Biderectional LSTM Spam detection
    ↪  architecture
308 model2.add(Embedding(VOCAB_SIZE, EMBEDING_DIM,
    ↪  input_length=MAX_LEN))
309 model2.add(Bidirectional(LSTM(N_LTSM, dropout=DROP_VALUE,
    ↪  return_sequences=True)))
310 model2.add(GlobalAveragePooling1D())
311 model2.add(Dense(1, activation='sigmoid'))
312 model2.compile(loss = 'binary_crossentropy', optimizer = 'adam',
    ↪  metrics=['accuracy'])
313 model2.summary()
314
315 plot_model(model2, to_file="m2.png", show_shapes=True,
    ↪  show_layer_names=True)
316
317 early_stop = EarlyStopping(monitor='val_loss', patience=3)
318 history2 = model2.fit(training_padded, train_labels,
    ↪  epochs=num_epochs, validation_data=(testing_padded,
    ↪  test_labels), callbacks = (modelcheckpoint, early_stop),
    ↪  verbose=1)
```

```
319
320 metrics = pd.DataFrame(history2.history)        # Create a
     ↪ dataframe
321 metrics.rename(columns = {'loss': 'Training_Loss', 'accuracy':
     ↪ 'Training_Accuracy', 'val_loss': 'Validation_Loss',
     ↪ 'val_accuracy': 'Validation_Accuracy'}, inplace = True)
322 def plot_graphs1(var1, var2, string):
323     metrics[[var1, var2]].plot()
324     plt.title('BiLSTM Model: Training and Validation ' + string)
325     plt.xlabel ('Number of epochs')
326     plt.ylabel(string)
327     plt.legend([var1, var2])
328 plot_graphs1('Training_Loss', 'Validation_Loss', 'loss')
329 plot_graphs1('Training_Accuracy', 'Validation_Accuracy',
     ↪ 'accuracy')
330
331 epochs = 30   # default is 4, 30 I use early stop for ending
332 batch_size = 128
333 max_review_length = 400         # vector-space embedding:
334 pad_type = trunc_type = 'pre'
335 N_CONV = 256 # filters, a.k.a. kernels         # convolutional
     ↪ layer architecture:
336 k_conv = 3 # kernel length
337 N_DENSE = 256        # dense layer architecture:
338
339 model3 = Sequential()        # deep-learning-illustrated-
     ↪ master/notebooks/convolutional_sentiment_classifier.ipynb
340 model3.add(Embedding(VOCAB_SIZE, EMBEDING_DIM,
     ↪ input_length=MAX_LEN))
341 model3.add(SpatialDropout1D(DROP_VALUE))
342 model3.add(Conv1D(N_CONV, k_conv, activation='relu'))
343 model3.add(Conv1D(N_CONV, k_conv, activation='relu'))
344 model3.add(GlobalMaxPooling1D())
345 model3.add(Dense(N_DENSE, activation='relu'))
346 model3.add(Dropout(DROP_VALUE))
347 model3.add(Dense(1, activation='sigmoid'))
348
349 model3.summary()
350
```

```
351 plot_model(model3, to_file="m3.png", show_shapes=True,
     ↪  show_layer_names=True)
352
353 model3.compile(loss='binary_crossentropy', optimizer='adam',
     ↪  metrics=['accuracy'])
354 modelcheckpoint = ModelCheckpoint(filepath = output_dir3 +
     ↪  "/weights.{epoch:02d}.hdf5")
355 if not os.path.exists(output_dir3):
356     os.makedirs(output_dir3)
357
358 early_stop = EarlyStopping(monitor='val_loss', patience=3)
359 history3 = model3.fit(training_padded, train_labels,
     ↪  batch_size=batch_size, epochs=epochs,
     ↪  validation_data=(testing_padded, test_labels), callbacks =
     ↪  (modelcheckpoint, early_stop), verbose=1)
360
361 metrics = pd.DataFrame(history3.history)        # Create a
     ↪  dataframe
362 metrics.rename(columns = {'loss': 'Training_Loss', 'accuracy':
     ↪  'Training_Accuracy', 'val_loss': 'Validation_Loss',
     ↪  'val_accuracy': 'Validation_Accuracy'}, inplace = True)
363 def plot_graphs3(var1, var2, string):
364     metrics[[var1, var2]].plot()
365     plt.title('Model: Training and Validation ' + string)
366     plt.xlabel ('Number of epochs')
367     plt.ylabel(string)
368     plt.legend([var1, var2])
369
370 plot_graphs3('Training_Loss', 'Validation_Loss', 'loss')
371 plot_graphs3('Training_Accuracy', 'Validation_Accuracy',
     ↪  'accuracy')
372
373 epochs = 30  # convolutional layer architecture:
374 batch_size = 128
375 pad_type = trunc_type = 'pre'         # vector-space embedding:
376 drop_embed = 0.2
377 n_conv_1 = n_conv_2 = n_conv_3 = 256
378 k_conv_1 = 3
379 k_conv_2 = 2
```

```
380 k_conv_3 = 4
381 n_dense = 256          # dense layer architecture:
382 dropout = 0.2
383 input_layer = Input(shape=(MAX_LEN,), dtype='int16',
     ↪ name='input')
384 embedding_layer = Embedding(VOCAB_SIZE, EMBEDING_DIM,
     ↪ name='embedding')(input_layer)
385 drop_embed_layer = SpatialDropout1D(drop_embed,
     ↪ name='drop_embed')(embedding_layer)
386 # three parallel convolutional streams:
387 conv_1 = Conv1D(n_conv_1, k_conv_1, activation='relu',
     ↪ name='conv_1')(drop_embed_layer)
388 maxp_1 = GlobalMaxPooling1D(name='maxp_1')(conv_1)
389 conv_2 = Conv1D(n_conv_2, k_conv_2, activation='relu',
     ↪ name='conv_2')(drop_embed_layer)
390 maxp_2 = GlobalMaxPooling1D(name='maxp_2')(conv_2)
391 conv_3 = Conv1D(n_conv_3, k_conv_3, activation='relu',
     ↪ name='conv_3')(drop_embed_layer)
392 maxp_3 = GlobalMaxPooling1D(name='maxp_3')(conv_3)
393 concat = concatenate([maxp_1, maxp_2, maxp_3])
394 dense_layer = Dense(n_dense, activation='relu',
     ↪ name='dense')(concat)
395 drop_dense_layer = Dropout(dropout,
     ↪ name='drop_dense')(dense_layer)
396 dense_2 = Dense(int(n_dense/4), activation='relu',
     ↪ name='dense_2')(drop_dense_layer)
397 dropout_2 = Dropout(dropout, name='drop_dense_2')(dense_2)
398 predictions = Dense(1, activation='sigmoid',
     ↪ name='output')(dropout_2)         # sigmoid output layer:
399 model4 = Model(input_layer, predictions)        # create model:
400 
401 model4.summary()
402 
403 plot_model(model4, to_file="m4.png", show_shapes=True,
     ↪ show_layer_names=True)
404 
405 model4.compile(loss='binary_crossentropy', optimizer='adam',
     ↪ metrics=['accuracy'])
406 output_dir4 = "model_output/multiconv" + msglength + version +
     ↪ lang
```

```
407 modelcheckpoint = ModelCheckpoint(filepath = output_dir4
    ↪ +"/weights.{epoch:02d}.hdf5")
408 if not os.path.exists(output_dir4):
409     os.makedirs(output_dir4)
410 early_stop = EarlyStopping(monitor='val_loss', patience=3)
411 history4 = model4.fit(training_padded, train_labels,
    ↪ batch_size=batch_size, epochs=epochs,
    ↪ validation_data=(testing_padded, test_labels), callbacks =
    ↪ (modelcheckpoint, early_stop), verbose=1)
412
413 metrics = pd.DataFrame(history4.history)          # Create a
    ↪ dataframe
414 metrics.rename(columns = {'loss': 'Training_Loss', 'accuracy':
    ↪ 'Training_Accuracy','val_loss': 'Validation_Loss',
    ↪ 'val_accuracy': 'Validation_Accuracy'}, inplace = True)
415 def plot_graphs4(var1, var2, string):
416     metrics[[var1, var2]].plot()
417     plt.title('Model: Training and Validation ' + string)
418     plt.xlabel ('Number of epochs')
419     plt.ylabel(string)
420     plt.legend([var1, var2])
421 plot_graphs4('Training_Loss', 'Validation_Loss', 'loss')
422 plot_graphs4('Training_Accuracy', 'Validation_Accuracy',
    ↪ 'accuracy')
423
424 epochs = 30   # convolutional layer architecture:
425 batch_size = 128
426 pad_type = trunc_type = 'pre'          # vector-space embedding:
427 drop_embed = 0.2
428 n_conv = 768
429 k_conv_1 = 1
430 k_conv_2 = 2
431 k_conv_3 = 3
432 k_conv_4 = 5
433 k_conv_5 = 7
434 k_conv_6 = 9
435 n_dense = 512          # dense layer architecture:
436 dropout = 0.2
437
```

```
438 input_layer = Input(shape=(MAX_LEN,),
439                     dtype='int16', name='input')
440 embedding_layer = Embedding(VOCAB_SIZE, EMBEDING_DIM,
    ↪ name='embedding')(input_layer)         # embedding:
441 drop_embed_layer = SpatialDropout1D(drop_embed,
    ↪ name='drop_embed')(embedding_layer)
442 conv_1 = Conv1D(n_conv, k_conv_1, activation='relu',
    ↪ name='conv_1')(drop_embed_layer)         # three parallel
    ↪ convolutional streams:
443 maxp_1 = GlobalMaxPooling1D(name='maxp_1')(conv_1)
444 conv_2 = Conv1D(n_conv, k_conv_2, activation='relu',
    ↪ name='conv_2')(drop_embed_layer)
445 maxp_2 = GlobalMaxPooling1D(name='maxp_2')(conv_2)
446 conv_3 = Conv1D(n_conv, k_conv_3,activation='relu',
    ↪ name='conv_3')(drop_embed_layer)
447 maxp_3 = GlobalMaxPooling1D(name='maxp_3')(conv_3)
448 conv_4 = Conv1D(n_conv, k_conv_4, activation='relu',
    ↪ name='conv_4')(drop_embed_layer)
449 maxp_4 = GlobalMaxPooling1D(name='maxp_4')(conv_4)
450 conv_5 = Conv1D(n_conv, k_conv_5, activation='relu',
    ↪ name='conv_5')(drop_embed_layer)
451 maxp_5 = GlobalMaxPooling1D(name='maxp_5')(conv_5)
452 conv_6 = Conv1D(n_conv, k_conv_6, activation='relu',
    ↪ name='conv_6')(drop_embed_layer)
453 maxp_6 = GlobalMaxPooling1D(name='maxp_6')(conv_6)
454 concat12 = concatenate([maxp_1, maxp_2])
455 concat34 = concatenate([maxp_3, maxp_4])
456 concat56 = concatenate([maxp_5, maxp_6])
457 concat = concatenate([concat12, concat34, concat56])
458 dense_layer = Dense(n_dense, activation='relu',
    ↪ name='dense')(concat)         # dense hidden layers:
459 drop_dense_layer = Dropout(dropout,
    ↪ name='drop_dense')(dense_layer)
460 dense_2 = Dense(int(n_dense/4), activation='relu',
    ↪ name='dense_2')(drop_dense_layer)
461 dropout_2 = Dropout(dropout, name='drop_dense_2')(dense_2)
462 predictions = Dense(1, activation='sigmoid',
    ↪ name='output')(dropout_2)         # sigmoid output layer:
463 model5 = Model(input_layer, predictions)         # create model
```

```
464
465 model5.summary()
466
467 plot_model(model5, to_file="m5.png", show_shapes=True,
    ↪ show_layer_names=True)
468
469 model5.compile(loss='binary_crossentropy', optimizer='adam',
    ↪ metrics=['accuracy'])
470 output_dir5 = "model_output/multiconv5" + msglength + version +
    ↪ lang
471 modelcheckpoint = ModelCheckpoint(filepath = output_dir5
    ↪ +"/weights.{epoch:02d}.hdf5")
472 if not os.path.exists(output_dir5):
473     os.makedirs(output_dir5)
474 history5 = model5.fit(training_padded, train_labels,
    ↪ batch_size=batch_size, epochs=epochs,
    ↪ validation_data=(testing_padded, test_labels), callbacks =
    ↪ (modelcheckpoint, early_stop), verbose=1)
475
476 metrics = pd.DataFrame(history5.history)        # Create a
    ↪ dataframe
477 metrics.rename(columns = {'loss': 'Training_Loss', 'accuracy':
    ↪ 'Training_Accuracy','val_loss': 'Validation_Loss',
    ↪ 'val_accuracy': 'Validation_Accuracy'}, inplace = True)
478 def plot_graphs5(var1, var2, string):
479     metrics[[var1, var2]].plot()
480     plt.title('Model: Training and Validation ' + string)
481     plt.xlabel ('Number of epochs')
482     plt.ylabel(string)
483     plt.legend([var1, var2])
484 plot_graphs5('Training_Loss', 'Validation_Loss', 'loss')
485 plot_graphs5('Training_Accuracy', 'Validation_Accuracy',
    ↪ 'accuracy')
486
487 # Comparing different models values
488 print(f"Dense architecture loss and accuracy:
    ↪ {model.evaluate(testing_padded, test_labels)} " )
489 print("**************************************")
490 print(f"LSTM architecture loss and accuracy:
    ↪ {model1.evaluate(testing_padded, test_labels)} " )
```

```
491 print("**************************************")
492 print(f"Bi-LSTM architecture loss and accuracy:
    ↪  {model2.evaluate(testing_padded, test_labels)} " )
493 print("**************************************")
494 print(f"Convolutional architecture loss and accuracy:
    ↪  {model3.evaluate(testing_padded, test_labels)} " )
495 print("**************************************")
496 print(f"Multi-ConvNet architecture loss and accuracy:
    ↪  {model4.evaluate(testing_padded, test_labels)} " )
497 print("**************************************")
498 print(f"Multi-ConvNet 5 architecture loss and accuracy:
    ↪  {model5.evaluate(testing_padded, test_labels)} " )
499 
500 # Comparing different models Graph
501 model.load_weights(output_dir + "/weights.03.hdf5")
502 y_hat = model.predict(testing_padded)
503 plt.hist(y_hat)
504 _ = plt.axvline(x=0.5, color='orange')
505 
506 "{:0.2f}".format(roc_auc_score(test_labels, y_hat)*100.0)
507 y_hat = np.array([float(i) for i in y_hat])
508 y_hat = np.array([round(i) for i in y_hat])
509 accuracy = accuracy_score(test_labels, y_hat)
510 "{:0.2f}".format((accuracy)*100.0)
511 
512 model1.load_weights(output_dir1 + "/weights.03.hdf5")
513 y_hat = model1.predict(testing_padded)
514 plt.hist(y_hat)
515 _ = plt.axvline(x=0.5, color='orange')
516 
517 "{:0.2f}".format(roc_auc_score(test_labels, y_hat)*100.0)
518 y_hat = np.array([float(i) for i in y_hat])
519 y_hat = np.array([round(i) for i in y_hat])
520 accuracy = accuracy_score(test_labels, y_hat)
521 "{:0.2f}".format((accuracy)*100.0)
522 
523 
524 model2.load_weights(output_dir2 + "/weights.03.hdf5")
525 y_hat = model2.predict(testing_padded)
```

```
526 plt.hist(y_hat)
527 _ = plt.axvline(x=0.5, color='orange')
528 
529 "{:0.2f}".format(roc_auc_score(test_labels, y_hat)*100.0)
530 y_hat = np.array([float(i) for i in y_hat])
531 y_hat = np.array([round(i) for i in y_hat])
532 accuracy = accuracy_score(test_labels, y_hat)
533 "{:0.2f}".format((accuracy)*100.0)
534 
535 model3.load_weights(output_dir3 + "/weights.03.hdf5")
536 y_hat = model3.predict(testing_padded)
537 plt.hist(y_hat)
538 _ = plt.axvline(x=0.5, color='orange')
539 
540 "{:0.2f}".format(roc_auc_score(test_labels, y_hat)*100.0)
541 y_hat = np.array([float(i) for i in y_hat])
542 y_hat = np.array([round(i) for i in y_hat])
543 accuracy = accuracy_score(test_labels, y_hat)
544 "{:0.2f}".format((accuracy)*100.0)
545 
546 model4.load_weights(output_dir4 + "/weights.03.hdf5")
547 y_hat = model4.predict(testing_padded)
548 plt.hist(y_hat)
549 _ = plt.axvline(x=0.5, color='orange')
550 
551 "{:0.2f}".format(roc_auc_score(test_labels, y_hat)*100.0)
552 y_hat = np.array([float(i) for i in y_hat])
553 y_hat = np.array([round(i) for i in y_hat])
554 accuracy = accuracy_score(test_labels, y_hat)
555 "{:0.2f}".format((accuracy)*100.0)
556 
557 model.load_weights(output_dir + "/weights.03.hdf5")
558 y_hat = model.predict(testing_padded)
559 plt.hist(y_hat)
560 _ = plt.axvline(x=0.5, color='orange')
561 
562 "{:0.2f}".format(roc_auc_score(test_labels, y_hat)*100.0)
563 y_hat = np.array([float(i) for i in y_hat])
564 y_hat = np.array([round(i) for i in y_hat])
```

```
565 accuracy = accuracy_score(test_labels, y_hat)
566 "{:0.2f}".format((accuracy)*100.0)
567
568 model5.load_weights(output_dir5 + "/weights.03.hdf5")
569 y_hat = model5.predict(testing_padded)
570 plt.hist(y_hat)
571 _ = plt.axvline(x=0.5, color='orange')
572
573 "{:0.2f}".format(roc_auc_score(test_labels, y_hat)*100.0)
574 y_hat = np.array([float(i) for i in y_hat])
575 y_hat = np.array([round(i) for i in y_hat])
576 accuracy = accuracy_score(test_labels, y_hat)
577 "{:0.2f}".format((accuracy)*100.0)
```

Listing 12: antispamSpacy.ipynb Führt die Daten aus dem Mail-Datensatz über Stemmer- und Lemma-Algorithmus auf den Wortstamm zurück.

A.5.4. labelCSVmailArgosTranslate.ipynb

```
1  # *Soll
2  # ** die Mails im CSV-Format einlesen
3  # ** die Laenge und die 3 wahrscheinlichsten Sprache erkennen
4  # ** die Mails ubersetzten
5  # ** Ausgabe der alten und neuen Daten in OriginalnameLabeled.csv
6
7  import numpy as np
8  import pandas as pd
9  import os # Reading and writing from Disk
10 import spacy # saetzte aufteilen
11 import fasttext  # # Language Detection
12 import argostranslate.package # Translation
13 import argostranslate.translate # Translation
14
15 filename = "1Monat18042024.csv"
```

```
16  messages = pd.read_csv(filename, sep =',',names=["label",
    ↪  "message"], header = 1)
17  PRETRAINED_MODEL_PATH = 'lid.176.bin'  # fastTextIstDochSpam.csv
18  modelLang = fasttext.load_model(PRETRAINED_MODEL_PATH)
19  print(messages.head())
20
21  type(messages)
22  messages.info()
23  messages.shape
24  messages.size
25  messages.dtypes
26  messages.columns
27
28  messages=messages.drop_duplicates()
29  messages.shape
30
31  print(messages.head())
32  avglength = sum(messages.message.str.len())/messages.size
33  allength = messages.message.str.len()
34  allength.describe(percentiles=[.1,.2,.3,.4,.5,.6,.7,.8,.9])
35
36  # load argostranslate models
37  argostranslate.package.install_from_path("argostranslate/translate-
    ↪  ar_en-1_0.argosmodel")
38  argostranslate.package.install_from_path("argostranslate/translate-
    ↪  bg_en-1_9.argosmodel")
39  argostranslate.package.install_from_path("argostranslate/translate-
    ↪  bn_en-1_9.argosmodel")
40  argostranslate.package.install_from_path("argostranslate/translate-
    ↪  ca_en-1_7.argosmodel")
41  argostranslate.package.install_from_path("argostranslate/translate-
    ↪  cs_en-1_9.argosmodel")
42  argostranslate.package.install_from_path("argostranslate/translate-
    ↪  da_en-1_3.argosmodel")
43  argostranslate.package.install_from_path("argostranslate/translate-
    ↪  de_en-1_0.argosmodel")
44  argostranslate.package.install_from_path("argostranslate/translate-
    ↪  el_en-1_9.argosmodel")
45  argostranslate.package.install_from_path("argostranslate/translate-
    ↪  eo_en-1_5.argosmodel")
```

```
46 argostranslate.package.install_from_path("argostranslate/translate-
   ↪ es_en-1_0.argosmodel")
47 argostranslate.package.install_from_path("argostranslate/translate-
   ↪ eu_en-1_9.argosmodel")
48 argostranslate.package.install_from_path("argostranslate/translate-
   ↪ fa_en-1_5.argosmodel")
49 argostranslate.package.install_from_path("argostranslate/translate-
   ↪ fi_en-1_5.argosmodel")
50 argostranslate.package.install_from_path("argostranslate/translate-
   ↪ fr_en-1_9.argosmodel")
51 argostranslate.package.install_from_path("argostranslate/translate-
   ↪ ga_en-1_1.argosmodel")
52 argostranslate.package.install_from_path("argostranslate/translate-
   ↪ hi_en-1_1.argosmodel")
53 argostranslate.package.install_from_path("argostranslate/translate-
   ↪ he_en-1_5.argosmodel")
54 argostranslate.package.install_from_path("argostranslate/translate-
   ↪ it_en-1_0.argosmodel")
55 argostranslate.package.install_from_path("argostranslate/translate-
   ↪ ja_en-1_1.argosmodel")
56 argostranslate.package.install_from_path("argostranslate/translate-
   ↪ lt_en-1_9.argosmodel")
57 argostranslate.package.install_from_path("argostranslate/translate-
   ↪ lv_en-1_9.argosmodel")
58 argostranslate.package.install_from_path("argostranslate/translate-
   ↪ ms_en-1_9.argosmodel")
59 argostranslate.package.install_from_path("argostranslate/translate-
   ↪ nb_en-1_9.argosmodel")
60 argostranslate.package.install_from_path("argostranslate/translate-
   ↪ nl_en-1_8.argosmodel")
61 argostranslate.package.install_from_path("argostranslate/translate-
   ↪ pl_en-1_9.argosmodel")
62 argostranslate.package.install_from_path("argostranslate/translate-
   ↪ pt_en-1_0.argosmodel")
63 argostranslate.package.install_from_path("argostranslate/translate-
   ↪ ro_en-1_9.argosmodel")
64 argostranslate.package.install_from_path("argostranslate/translate-
   ↪ ru_en-1_9.argosmodel")
65 argostranslate.package.install_from_path("argostranslate/translate-
   ↪ sk_en-1_5.argosmodel")
```

```
66 argostranslate.package.install_from_path("argostranslate/translate-
   ↪  sl_en-1_9.argosmodel")
67 argostranslate.package.install_from_path("argostranslate/translate-
   ↪  sv_en-1_5.argosmodel")
68 argostranslate.package.install_from_path("argostranslate/translate-
   ↪  th_en-1_9.argosmodel")
69 argostranslate.package.install_from_path("argostranslate/translate-
   ↪  tl_en-1_9.argosmodel")
70 argostranslate.package.install_from_path("argostranslate/translate-
   ↪  tr_en-1_5.argosmodel")
71 argostranslate.package.install_from_path("argostranslate/translate-
   ↪  uk_en-1_4.argosmodel")
72 argostranslate.package.install_from_path("argostranslate/translate-
   ↪  ur_en-1_9.argosmodel")
73
74 #spacy.require_gpu() # faster without for the small models
75 source_lang = spacy.load("xx_sent_ud_sm") # ['senter']
76
77 get_ipython().system('date') # start of translation time
78
79 langList = []
80 langProp = []
81 langList2 = []
82 langProp2 = []
83 langList3 = []
84 langProp3 = []
85 lenList = []
86 translatedList = []
87 for msg in messages['message']:
88         pred = modelLang.predict(str(msg),k=3)
89         print(pred)
90         zlang = len(pred[0])
91         if(zlang == 1):
92             lang = pred [0][0]
93             prop = pred [1][0]
94             prop2 = prop
95             lang2 = lang
96             prop3 = prop
97             lang3 = lang
```

```
98          if(zlang == 2):
99              lang = pred [0][0]
100             prop = pred [1][0]
101             prop2 = pred[1][1]
102             lang2 = pred[0][1]
103             prop3 = prop2
104             lang3 = lang2
105         if(zlang == 3):
106             lang = pred [0][0]
107             prop = pred [1][0]
108             prop2 = pred[1][1]
109             lang2 = pred[0][1]
110             lang3 = pred[0][2]
111             prop3 = pred[1][2]
112 
113         #print(lang,prop, lang2)
114         langList.append(lang)
115         langProp.append(prop)
116         langList2.append(lang2)
117         langProp2.append(prop2)
118         langList3.append(lang3)
119         langProp3.append(prop3)
120         doc = source_lang(str(msg)) # Process the source text
121         translated_text = ""  # Initialize the translated text
             ↪  variable
122         newText = ""
123         for sent in doc.sents: # Iterate over sentences in the
             ↪  processed text
124             if(str(lang) == "__label__ar"):
125                 newText =
                     ↪  argostranslate.translate.translate(sent.text,
                     ↪  "ar", "en") + " "
126             elif(str(lang) == "__label__bg"):
127                 newText =
                     ↪  argostranslate.translate.translate(sent.text,
                     ↪  "bg", "en") + " "
128             elif(str(lang) == "__label__bn"):
129                 newText =
                     ↪  argostranslate.translate.translate(sent.text,
                     ↪  "bn", "en") + " "
```

```
130                 elif(str(lang) == "__label__ca"):
131                     newText =
                         ↪  argostranslate.translate.translate(sent.text,
                         ↪  "ca", "en") + " "
132                 elif(str(lang) == "__label__cs"):
133                     newText =
                         ↪  argostranslate.translate.translate(sent.text,
                         ↪  "cs", "en") + " "
134                 elif(str(lang) == "__label__da"):
135                     newText =
                         ↪  argostranslate.translate.translate(sent.text,
                         ↪  "da", "en") + " "
136                 elif(str(lang) == "__label__de"):
137                     newText =
                         ↪  argostranslate.translate.translate(sent.text,
                         ↪  "de", "en") + " "
138                 elif(str(lang) == "__label__el"):
139                     newText =
                         ↪  argostranslate.translate.translate(sent.text,
                         ↪  "el", "en") + " "
140                 elif(str(lang) == "__label__en"):
141                     newText = sent.text + " "
142                 elif(str(lang) == "__label__eo"):
143                     newText =
                         ↪  argostranslate.translate.translate(sent.text,
                         ↪  "eo", "en") + " "
144                 elif(str(lang) == "__label__es"):
145                     newText =
                         ↪  argostranslate.translate.translate(sent.text,
                         ↪  "es", "en") + " "
146                 elif(str(lang) == "__label__eu"):
147                     newText =
                         ↪  argostranslate.translate.translate(sent.text,
                         ↪  "eu", "en") + " "
148                 elif(str(lang) == "__label__fa"):
149                     newText =
                         ↪  argostranslate.translate.translate(sent.text,
                         ↪  "fa", "en") + " "
150                 elif(str(lang) == "__label__fi"):
```

```
151                     newText =
                         ↪ argostranslate.translate.translate(sent.text,
                         ↪ "fi", "en") + " "
152                 elif(str(lang) == "__label__fr"):
153                     newText =
                         ↪ argostranslate.translate.translate(sent.text,
                         ↪ "fr", "en") + " "
154                 elif(str(lang) == "__label__ga"):
155                     newText =
                         ↪ argostranslate.translate.translate(sent.text,
                         ↪ "ga", "en") + " "
156                 elif(str(lang) == "__label__he"):
157                     newText =
                         ↪ argostranslate.translate.translate(sent.text,
                         ↪ "he", "en") + " "
158                 elif(str(lang) == "__label__hi"):
159                     newText =
                         ↪ argostranslate.translate.translate(sent.text,
                         ↪ "hi", "en") + " "
160                 elif(str(lang) == "__label__it"):
161                     newText =
                         ↪ argostranslate.translate.translate(sent.text,
                         ↪ "it", "en") + " "
162                 elif(str(lang) == "__label__ja"):
163                     newText =
                         ↪ argostranslate.translate.translate(sent.text,
                         ↪ "ja", "en") + " "
164                 elif(str(lang) == "__label__lt"):
165                     newText =
                         ↪ argostranslate.translate.translate(sent.text,
                         ↪ "lt", "en") + " "
166                 elif(str(lang) == "__label__lv"):
167                     newText =
                         ↪ argostranslate.translate.translate(sent.text,
                         ↪ "lv", "en") + " "
168                 elif(str(lang) == "__label__ms"):
169                     newText =
                         ↪ argostranslate.translate.translate(sent.text,
                         ↪ "ms", "en") + " "
```

```
170                 elif(str(lang) == "__label__nb"):
171                     newText =
                         ↪ argostranslate.translate.translate(sent.text,
                         ↪ "nb", "en") + " "
172                 elif(str(lang) == "__label__nl"):
173                     newText =
                         ↪ argostranslate.translate.translate(sent.text,
                         ↪ "nl", "en") + " "
174                 elif(str(lang) == "__label__pl"):
175                     newText =
                         ↪ argostranslate.translate.translate(sent.text,
                         ↪ "pl", "en") + " "
176                 elif(str(lang) == "__label__pt"):
177                     newText =
                         ↪ argostranslate.translate.translate(sent.text,
                         ↪ "pt", "en") + " "
178                 elif(str(lang) == "__label__ro"):
179                     newText =
                         ↪ argostranslate.translate.translate(sent.text,
                         ↪ "ro", "en") + " "
180                 elif(str(lang) == "__label__ru"):
181                     newText =
                         ↪ argostranslate.translate.translate(sent.text,
                         ↪ "ru", "en") + " "
182                 elif(str(lang) == "__label__sk"):
183                     newText =
                         ↪ argostranslate.translate.translate(sent.text,
                         ↪ "sk", "en") + " "
184                 elif(str(lang) == "__label__sl"):
185                     newText =
                         ↪ argostranslate.translate.translate(sent.text,
                         ↪ "sl", "en") + " "
186                 elif(str(lang) == "__label__sv"):
187                     newText =
                         ↪ argostranslate.translate.translate(sent.text,
                         ↪ "sv", "en") + " "
188                 elif(str(lang) == "__label__th"):
189                     newText =
                         ↪ argostranslate.translate.translate(sent.text,
                         ↪ "th", "en") + " "
```

```
190                 elif(str(lang) == "__label__tl"):
191                     newText =
                         ↪  argostranslate.translate.translate(sent.text,
                         ↪  "tl", "en") + " "
192                 elif(str(lang) == "__label__tr"):
193                     newText =
                         ↪  argostranslate.translate.translate(sent.text,
                         ↪  "tr", "en") + " "
194                 elif(str(lang) == "__label__uk"):
195                     newText =
                         ↪  argostranslate.translate.translate(sent.text,
                         ↪  "uk", "en") + " "
196                 elif(str(lang) == "__label__ur"):
197                     newText =
                         ↪  argostranslate.translate.translate(sent.text,
                         ↪  "ur", "en") + " "
198                 else: # not supported
199                     newText = sent.text + " "
200                 translated_text = translated_text + newText  + " "
201
             ↪  print("------------------------------------------------------------
202             translatedList.append(translated_text)
203             if(len(msg) <= 1000):
204                 lenList.append("short")
205             elif(len(msg) < 2500):
206                 lenList.append("middle")
207             else:
208                 lenList.append("long")
209
210 get_ipython().system('date')        # end of translation time
211
212 messages["lang"] = langList
213 messages["prop"] = langProp
214 messages["lang2"] = langList2
215 messages["prop2"] = langProp2
216 messages["lang3"] = langList3
217 messages["prop3"] = langProp3
218 messages["len"] = lenList
219 messages["translated"] = translatedList
```

```
220 print(messages.head())
221
222 messages.describe().T
223
224 messages.groupby('label').describe()
225
226 messages.groupby('lang').describe()
227
228 messages['lang'].value_counts().head(20) # lang most prop.
229
230 messages['prop'].value_counts().head(20)
231
232 messages['lang2'].value_counts().head(20) # lang second prop.
233
234 messages['prop2'].value_counts().head(20)
235
236 messages['lang3'].value_counts().head(20) # lang third prop.
237
238 messages['prop3'].value_counts().head(20)
239
240 messages.groupby('len').describe()
241
242 outputFilename = "lab3_" + filename
243 messages.to_csv(outputFilename)
```

Listing 13: labelCSVmailArgosTranslate.ipynb Übersetzt den Mail-Datensatz auf Englisch.

A.6. Training und Testsklassische NN

A.6.1. antispam.ipynb

```
1 import tensorflow as tf
2 print(tf.__version__)
3 print("Num GPUs Available: ",
  ↪ len(tf.config.list_physical_devices('GPU')))
```

```
4
5  import pickle   # Load Save Tokenizer
6  import numpy as np
7  import pandas as pd
8  import seaborn as sns
9  import matplotlib.pyplot as plt
10 from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
11 %matplotlib inline
12
13 from sklearn.model_selection import train_test_split # library
    ↪  for train test split
14 from sklearn.metrics import roc_auc_score # for roc_auc_score
15 from sklearn.metrics import log_loss, accuracy_score
16 # Modeling
17 import tensorflow as tf # deep learning libraries for text
    ↪  pre-processing
18 from tensorflow.keras.preprocessing.text import Tokenizer
19 from tensorflow.keras.preprocessing.sequence import pad_sequences
20 from tensorflow.keras.callbacks import EarlyStopping
21 from tensorflow.keras.models import Sequential
22 from tensorflow.keras.layers import Embedding,
    ↪  GlobalAveragePooling1D,  Flatten, Dense, Dropout, Embedding,
    ↪  LSTM, SpatialDropout1D, Bidirectional, SpatialDropout1D,
    ↪  Conv1D, GlobalMaxPooling1D
23 from keras.callbacks import ModelCheckpoint
24 from keras.layers import Input, concatenate # Multi-ConvNet
25 from keras.models import Model
26 import os # Reading and writing from Disk
27 from tensorflow.keras.utils import plot_model # For Graphic of
    ↪  the DNN
28
29 msglength = "msg"  ### every
30 #msglength = "short"
31 #msglength = "middle"
32 #msglength = "long"
33 version = "128" # Model version
34 #lang = "__label__de"
35 #lang = "__label__en"
36 #lang = "__label__somtehingElse"
```

```
37  lang = "all"
38
39  #filename = "lab_mail65plus.csv"
40  filename = "6jSpamHam1q24_v46en.csv"        # min 30GB Ram
41  # messages = pd.read_csv(filename, sep =',',names=["label",
     ↪  "message","lang","len"], header = 0)
42  messages = pd.read_csv(filename, sep =',' ,names =
     ↪  ["nr","label","message","lang","prop","len","nil"] , header =
     ↪  1)
43  MAX_LEN = 512 # 300 # pad_sequencs parameter, only look for X
     ↪  words in a sentence (around 80% of the sentence
44  VOCAB_SIZE = 32768 #15626   # need a fix size
45
46  type(messages)
47  messages.info()
48  messages.shape
49  messages.size
50  messages.dtypes
51  messages.columns
52
53  messages=messages.drop_duplicates()        # Delete duplicates
54  messages.shape
55
56  print(messages.head())        # How is the length of the messages
     ↪  in char distributed
57  avglength = sum(messages.message.str.len())/messages.size
58  allength = messages.message.str.len()
59  allength.describe(percentiles=[.1,.2,.3,.4,.5,.6,.7,.8,.9])
60
61  dropList = []
62  cnt = 0
63  if(msglength != "msg"):
64      for msg in messages['len']:
65          #print(msg)
66          if(msg != msglength):
67              dropList.append(cnt)
68          cnt = cnt + 1
69  print(dropList[0:4])
70
```

```
71  print(messages.index[0:10])
72
73  messages.drop(messages.index[dropList], inplace=True) # deletes
     ↪  not used messages
74  messages.shape
75
76  dropList = []
77  cnt = 0
78  if(lang != "all"): # Alle Sprachen zu machen
79      if(lang == "__label__de" or lang == "__label__en"): # lang is
         ↪  something else
80          for msg in messages['lang']:
81              #print(msg)
82              if(msg != lang):
83                      dropList.append(cnt)
84              cnt = cnt + 1
85      else:   # Language is de or en
86          for msg in messages['lang']:
87              #print(msg)
88              if(msg == "__label__de" or msg == "__label__en"):
89                      dropList.append(cnt)
90              cnt = cnt + 1
91
92  print(dropList)
93
94  messages.drop(messages.index[dropList], inplace=True) # deletes
     ↪  not used messages
95  messages.shape
96
97  messages.groupby('label').describe().T
98
99  # get all the ham and spam emails
100 ham_msg = messages[messages.label == 'ham']
101 spam_msg = messages[messages.label == 'spam']
102 # For ham and spam messages, create numpy list to visualize using
     ↪  wordcloud
103 ham_msg_text = " ".join(ham_msg.message.to_numpy().tolist())
104 spam_msg_text = " ".join(spam_msg.message.to_numpy().tolist())
105
```

```
106 # Percentage of spam messages
107 (len(spam_msg)/len(ham_msg))*100        # Percentage of spam
    ↪ messages
108
109 if(len(ham_msg) > len(spam_msg)):  # take only the newer ones.
    ↪ the file for HAM is sorted by date after the first year
110     ham_msg_df = ham_msg.sort_index() # Zeile sollte nich
        ↪ gebraucht werden denk ich
111     ham_msg_df = ham_msg_df.head(n=len(spam_msg))
112     spam_msg_df = spam_msg
113 else: # more spam then ham, take the first entrys of SPAM in this
    ↪ case
114     spam_msg_df = spam_msg.sort_index() # Zeile sollte nich
        ↪ gebraucht werden denk ich
115     spam_msg_df = spam_msg_df.head(n = len(ham_msg))
116     ham_msg_df = ham_msg
117 print(ham_msg_df.shape, spam_msg_df.shape)
118
119 msg_df = pd.concat([ham_msg_df,spam_msg_df]) # Create a dataframe
    ↪ with these ham and spam msg
120 print(msg_df.shape)
121
122 plt.figure(figsize=(8,6))
123 sns.countplot(msg_df.label)
124 plt.title('Distribution of ham and spam email messages (after
    ↪ downsampling)')
125 plt.xlabel('Message types')
126
127 # Compare the length of HAM and SPAM Messages
128 msg_df['text_length'] = msg_df['message'].apply(len)        # Get
    ↪ length column for each text
129 print(msg_df['text_length'])
130
131 msg_df['msg_type']= msg_df['label'].map({'ham': 0, 'spam': 1})
132 msg_label = msg_df['msg_type'].values
133
134 arr = msg_label
135 print("Array is of type: ", type(arr))        # Printing type of
    ↪ arr object
```

```
136 print("No. of dimensions: ", arr.ndim)          # Printing array
     ↪  dimensions (axes)
137 print("Shape of array: ", arr.shape)          # Printing shape of
     ↪  array
138 print("Size of array: ", arr.size)          # Printing size (total
     ↪  number of elements) of array
139 print("Array stores elements of type: ", arr.dtype)          #
     ↪  Printing type of elements in array
140 print(arr)
141
142 type(msg_df)
143
144 msg_df
145
146 wordsInMessages = []
147 for msg in msg_df['message']:          # Words in Message
148     wordsInMessages.append(len(msg.split()))
149 np_wordsInMessages = np.array(wordsInMessages)          # to numpy
     ↪  array
150 df_wordsInMessages = pd.DataFrame(np_wordsInMessages, columns =
     ↪  ['CntWords']) # to DataFrame
151 df_wordsInMessages.describe(percentiles=[.1,.2,.3,.4,.5,.6,.7,.8,.9])
152
153 train_msg, test_msg, train_labels, test_labels =
     ↪  train_test_split(msg_df['message'], msg_label, test_size=0.2,
     ↪  random_state=434)
154
155 trunc_type = "post" # pad_sequences parameter
156 padding_type = "post" # pad_sequences parameter
157 oov_tok = "<OOV>" # out of vocabulary token
158
159 tokenizer = Tokenizer(num_words = VOCAB_SIZE, char_level=False,
     ↪  oov_token = oov_tok)
160 tokenizer.fit_on_texts(train_msg)
161
162 word_index = tokenizer.word_index          # Get the word_index
163 word_index
164
165 tot_words = len(word_index)          # check how many words
```

```
166 print('There are %s unique tokens in training data. ' %
    ↪  tot_words)
167
168 tokenizer = Tokenizer(num_words = VOCAB_SIZE, char_level=False,
    ↪  oov_token = oov_tok)
169 tokenizer.fit_on_texts(train_msg)
170 output_dir = "model_output/" + msglength + version + lang +"/"
171 if not os.path.exists(output_dir):
172     os.makedirs(output_dir)
173 with open(output_dir + 'tokenizer.pickle', 'wb') as handle:
174     pickle.dump(tokenizer, handle,
        ↪  protocol=pickle.HIGHEST_PROTOCOL)
175
176 training_sequences =
    ↪  tokenizer.texts_to_sequences(train_msg)        # Sequencing
    ↪  and padding on training and testing
177 training_padded = pad_sequences (training_sequences, maxlen =
    ↪  MAX_LEN,
178                                  padding = padding_type,
                                      ↪  truncating = trunc_type )
179 testing_sequences = tokenizer.texts_to_sequences(test_msg)
180 testing_padded = pad_sequences(testing_sequences, maxlen =
    ↪  MAX_LEN,
181                               padding = padding_type, truncating
                                   ↪  = trunc_type)
182
183 print('Shape of training tensor: ',
    ↪  training_padded.shape)         # Shape of train tensor
184 print('Shape of testing tensor: ', testing_padded.shape)
185
186 print(training_sequences[0])
187
188 len(training_sequences[0]), len(training_sequences[1])         #
    ↪  Before padding : first sequence has a different length of the
    ↪  second
189
190 len(training_padded[0]), len(training_padded[1])         # Padded
    ↪  to same length of 50
191
```

```
192 print(training_padded[0])
193 print(training_sequences[0])
194 
195 # Dense sentiment model architecture
196 EMBEDING_DIM = 32   # 16 bei SMS
197 DROP_VALUE = 0.2 # dropout
198 N_DENSE = 24    #
199 model = Sequential()          #Dense sentiment model architecture
200 model.add(Embedding(VOCAB_SIZE, EMBEDING_DIM,
    ↪  input_length=MAX_LEN))
201 model.add(GlobalAveragePooling1D())
202 model.add(Dense(N_DENSE, activation='relu'))
203 model.add(Dropout(DROP_VALUE))
204 model.add(Dense(1, activation='sigmoid'))
205 model.summary()
206 
207 plot_model(model, to_file="m.png", show_shapes=True,
    ↪  show_layer_names=True)
208 
209 from tensorflow import keras
210 optim = keras.optimizers.Adam(learning_rate=0.00001)
211 output_dir = "model_output/dense0" + msglength + version + lang
212 modelcheckpoint = ModelCheckpoint(filepath = output_dir
    ↪  +"/weights.{epoch:02d}.hdf5")
213 if not os.path.exists(output_dir):
214     os.makedirs(output_dir)
215 model.compile(loss='binary_crossentropy',optimizer='adam'
    ↪  ,metrics=['accuracy'])
216 num_epochs = 30 # fitting a dense spam detector model
217 early_stop = EarlyStopping(monitor='val_loss', patience=5)
218 history = model.fit(training_padded, train_labels,
    ↪  epochs=num_epochs, validation_data=(testing_padded,
    ↪  test_labels),callbacks = (modelcheckpoint, early_stop),
    ↪  verbose=1)
219 
220 metrics = pd.DataFrame(history.history)
221 metrics[:2]
222 metrics.rename(columns = {'loss': 'Training_Loss', 'accuracy':
    ↪  'Training_Accuracy','val_loss': 'Validation_Loss',
    ↪  'val_accuracy': 'Validation_Accuracy'}, inplace = True)
```

```
223 metrics[:2]
224
225 def plot_graphs1(var1, var2, string):
226     metrics[[var1, var2]].plot()
227     plt.title('Dense Classifier: Training and Validation ' +
         ↪ string)
228     plt.xlabel ('Number of epochs')
229     plt.ylabel(string)
230     plt.legend([var1, var2])
231 plot_graphs1('Training_Loss', 'Validation_Loss', 'loss')
232 plot_graphs1('Training_Accuracy', 'Validation_Accuracy',
     ↪ 'accuracy')
233
234 N_LTSM = 20          #LSTM layer arcitecture hyperparameters
235 output_dir1 = "model_output/ltsm" + msglength + version + lang
236 modelcheckpoint = ModelCheckpoint(filepath = output_dir1 +
     ↪ "/weights.{epoch:02d}.hdf5")
237 if not os.path.exists(output_dir1):
238     os.makedirs(output_dir1)
239 model1 = Sequential()
240 model1.add(Embedding(VOCAB_SIZE, EMBEDING_DIM,
     ↪ input_length=MAX_LEN))
241 model1.add(LSTM(N_LTSM, dropout=DROP_VALUE,
     ↪ return_sequences=True))
242 model1.add(LSTM(N_LTSM, dropout=DROP_VALUE,
     ↪ return_sequences=True))
243 model1.add(GlobalAveragePooling1D())
244 model1.add(Dense(1, activation='sigmoid'))
245 model1.summary()
246
247 plot_model(model1, to_file="m1.png", show_shapes=True,
     ↪ show_layer_names=True)
248
249 model1.compile(loss = 'binary_crossentropy', optimizer = 'adam',
     ↪ metrics=['accuracy'])
250
251 num_epochs = 30
252 early_stop = EarlyStopping(monitor='val_loss', patience=3)
253 history1 = model1.fit(training_padded, train_labels,
     ↪ epochs=num_epochs, validation_data=(testing_padded,
     ↪ test_labels), callbacks = (modelcheckpoint, early_stop),
     ↪ verbose=1)
```

```
254
255 metrics = pd.DataFrame(history1.history)          # Create a
    ↪ dataframe
256 metrics.rename(columns = {'loss': 'Training_Loss', 'accuracy':
    ↪ 'Training_Accuracy', 'val_loss': 'Validation_Loss',
    ↪ 'val_accuracy': 'Validation_Accuracy'}, inplace = True)
257 def plot_graphs(var1, var2, string):
258     metrics[[var1, var2]].plot()
259     plt.title('LSTM Model: Training and Validation ' + string)
260     plt.xlabel ('Number of epochs')
261     plt.ylabel(string)
262     plt.legend([var1, var2])
263 plot_graphs('Training_Loss', 'Validation_Loss', 'loss')
264 plot_graphs('Training_Accuracy', 'Validation_Accuracy',
    ↪ 'accuracy')
265
266 output_dir2 = "model_output/bi-ltsm" + msglength + version + lang
267 modelcheckpoint = ModelCheckpoint(filepath = output_dir2 +
    ↪ "/weights.{epoch:02d}.hdf5")
268 if not os.path.exists(output_dir2):
269     os.makedirs(output_dir2)
270
271 model2 = Sequential()          # Biderectional LSTM Spam detection
    ↪ architecture
272 model2.add(Embedding(VOCAB_SIZE, EMBEDING_DIM,
    ↪ input_length=MAX_LEN))
273 model2.add(Bidirectional(LSTM(N_LTSM, dropout=DROP_VALUE,
    ↪ return_sequences=True)))
274 model2.add(GlobalAveragePooling1D())
275 model2.add(Dense(1, activation='sigmoid'))
276 model2.compile(loss = 'binary_crossentropy', optimizer = 'adam',
    ↪ metrics=['accuracy'])
277 model2.summary()
278
279 plot_model(model2, to_file="m2.png", show_shapes=True,
    ↪ show_layer_names=True)
280
281 early_stop = EarlyStopping(monitor='val_loss', patience=3)
282 history2 = model2.fit(training_padded, train_labels,
    ↪ epochs=num_epochs, validation_data=(testing_padded,
    ↪ test_labels), callbacks = (modelcheckpoint, early_stop),
    ↪ verbose=1)
```

```
283
284 metrics = pd.DataFrame(history2.history)         # Create a
    ↪  dataframe
285 metrics.rename(columns = {'loss': 'Training_Loss', 'accuracy':
    ↪  'Training_Accuracy', 'val_loss': 'Validation_Loss',
    ↪  'val_accuracy': 'Validation_Accuracy'}, inplace = True)
286 def plot_graphs1(var1, var2, string):
287     metrics[[var1, var2]].plot()
288     plt.title('BiLSTM Model: Training and Validation ' + string)
289     plt.xlabel ('Number of epochs')
290     plt.ylabel(string)
291     plt.legend([var1, var2])
292 plot_graphs1('Training_Loss', 'Validation_Loss', 'loss')
293 plot_graphs1('Training_Accuracy', 'Validation_Accuracy',
    ↪  'accuracy')
294
295 epochs = 30    # default is 4, 30 I use early stop for ending
296 batch_size = 128
297 max_review_length = 400          # vector-space embedding:
298 pad_type = trunc_type = 'pre'
299 N_CONV = 256 # filters, a.k.a. kernels         # convolutional
    ↪  layer architecture:
300 k_conv = 3 # kernel length
301 N_DENSE = 256         # dense layer architecture:
302
303 model3 = Sequential()         # deep-learning-illustrated-
    ↪  master/notebooks/convolutional_sentiment_classifier.ipynb
304 model3.add(Embedding(VOCAB_SIZE, EMBEDING_DIM,
    ↪  input_length=MAX_LEN))
305 model3.add(SpatialDropout1D(DROP_VALUE))
306 model3.add(Conv1D(N_CONV, k_conv, activation='relu'))
307 model3.add(Conv1D(N_CONV, k_conv, activation='relu'))
308 model3.add(GlobalMaxPooling1D())
309 model3.add(Dense(N_DENSE, activation='relu'))
310 model3.add(Dropout(DROP_VALUE))
311 model3.add(Dense(1, activation='sigmoid'))
312
313 model3.summary()
314
```

```
315 plot_model(model3, to_file="m3.png", show_shapes=True,
    ↪ show_layer_names=True)
316
317 model3.compile(loss='binary_crossentropy', optimizer='adam',
    ↪ metrics=['accuracy'])
318 modelcheckpoint = ModelCheckpoint(filepath = output_dir3 +
    ↪ "/weights.{epoch:02d}.hdf5")
319 if not os.path.exists(output_dir3):
320     os.makedirs(output_dir3)
321
322 early_stop = EarlyStopping(monitor='val_loss', patience=3)
323 history3 = model3.fit(training_padded, train_labels,
    ↪ batch_size=batch_size, epochs=epochs,
    ↪ validation_data=(testing_padded, test_labels), callbacks =
    ↪ (modelcheckpoint, early_stop), verbose=1)
324
325 metrics = pd.DataFrame(history3.history)          # Create a
    ↪ dataframe
326 metrics.rename(columns = {'loss': 'Training_Loss', 'accuracy':
    ↪ 'Training_Accuracy', 'val_loss': 'Validation_Loss',
    ↪ 'val_accuracy': 'Validation_Accuracy'}, inplace = True)
327 def plot_graphs3(var1, var2, string):
328     metrics[[var1, var2]].plot()
329     plt.title('Model: Training and Validation ' + string)
330     plt.xlabel ('Number of epochs')
331     plt.ylabel(string)
332     plt.legend([var1, var2])
333
334 plot_graphs3('Training_Loss', 'Validation_Loss', 'loss')
335 plot_graphs3('Training_Accuracy', 'Validation_Accuracy',
    ↪ 'accuracy')
336
337 epochs = 30  # convolutional layer architecture:
338 batch_size = 128
339 pad_type = trunc_type = 'pre'          # vector-space embedding:
340 drop_embed = 0.2
341 n_conv_1 = n_conv_2 = n_conv_3 = 256
342 k_conv_1 = 3
343 k_conv_2 = 2
```

```
344 k_conv_3 = 4
345 n_dense = 256          # dense layer architecture:
346 dropout = 0.2
347 input_layer = Input(shape=(MAX_LEN,), dtype='int16',
     ↪  name='input')
348 embedding_layer = Embedding(VOCAB_SIZE, EMBEDING_DIM,
     ↪  name='embedding')(input_layer)
349 drop_embed_layer = SpatialDropout1D(drop_embed,
     ↪  name='drop_embed')(embedding_layer)
350 # three parallel convolutional streams:
351 conv_1 = Conv1D(n_conv_1, k_conv_1, activation='relu',
     ↪  name='conv_1')(drop_embed_layer)
352 maxp_1 = GlobalMaxPooling1D(name='maxp_1')(conv_1)
353 conv_2 = Conv1D(n_conv_2, k_conv_2, activation='relu',
     ↪  name='conv_2')(drop_embed_layer)
354 maxp_2 = GlobalMaxPooling1D(name='maxp_2')(conv_2)
355 conv_3 = Conv1D(n_conv_3, k_conv_3, activation='relu',
     ↪  name='conv_3')(drop_embed_layer)
356 maxp_3 = GlobalMaxPooling1D(name='maxp_3')(conv_3)
357 concat = concatenate([maxp_1, maxp_2, maxp_3])
358 dense_layer = Dense(n_dense, activation='relu',
     ↪  name='dense')(concat)
359 drop_dense_layer = Dropout(dropout,
     ↪  name='drop_dense')(dense_layer)
360 dense_2 = Dense(int(n_dense/4), activation='relu',
     ↪  name='dense_2')(drop_dense_layer)
361 dropout_2 = Dropout(dropout, name='drop_dense_2')(dense_2)
362 predictions = Dense(1, activation='sigmoid',
     ↪  name='output')(dropout_2)        # sigmoid output layer:
363 model4 = Model(input_layer, predictions)        # create model:
364 
365 model4.summary()
366 
367 plot_model(model4, to_file="m4.png", show_shapes=True,
     ↪  show_layer_names=True)
368 
369 model4.compile(loss='binary_crossentropy', optimizer='adam',
     ↪  metrics=['accuracy'])
370 output_dir4 = "model_output/multiconv" + msglength + version +
     ↪  lang
```

```
371 modelcheckpoint = ModelCheckpoint(filepath = output_dir4
    ↪ +"/weights.{epoch:02d}.hdf5")
372 if not os.path.exists(output_dir4):
373     os.makedirs(output_dir4)
374 early_stop = EarlyStopping(monitor='val_loss', patience=3)
375 history4 = model4.fit(training_padded, train_labels,
    ↪ batch_size=batch_size, epochs=epochs,
    ↪ validation_data=(testing_padded, test_labels), callbacks =
    ↪ (modelcheckpoint, early_stop), verbose=1)
376
377 metrics = pd.DataFrame(history4.history)        # Create a
    ↪ dataframe
378 metrics.rename(columns = {'loss': 'Training_Loss', 'accuracy':
    ↪ 'Training_Accuracy','val_loss': 'Validation_Loss',
    ↪ 'val_accuracy': 'Validation_Accuracy'}, inplace = True)
379 def plot_graphs4(var1, var2, string):
380     metrics[[var1, var2]].plot()
381     plt.title('Model: Training and Validation ' + string)
382     plt.xlabel ('Number of epochs')
383     plt.ylabel(string)
384     plt.legend([var1, var2])
385 plot_graphs4('Training_Loss', 'Validation_Loss', 'loss')
386 plot_graphs4('Training_Accuracy', 'Validation_Accuracy',
    ↪ 'accuracy')
387
388 epochs = 30  # convolutional layer architecture:
389 batch_size = 128
390 pad_type = trunc_type = 'pre'        # vector-space embedding:
391 drop_embed = 0.2
392 n_conv = 768
393 k_conv_1 = 1
394 k_conv_2 = 2
395 k_conv_3 = 3
396 k_conv_4 = 5
397 k_conv_5 = 7
398 k_conv_6 = 9
399 n_dense = 512         # dense layer architecture:
400 dropout = 0.2
401
```

```
402 input_layer = Input(shape=(MAX_LEN,),
403                         dtype='int16', name='input')
404 embedding_layer = Embedding(VOCAB_SIZE, EMBEDING_DIM,
     ↪  name='embedding')(input_layer)          # embedding:
405 drop_embed_layer = SpatialDropout1D(drop_embed,
     ↪  name='drop_embed')(embedding_layer)
406 conv_1 = Conv1D(n_conv, k_conv_1, activation='relu',
     ↪  name='conv_1')(drop_embed_layer)          # three parallel
     ↪  convolutional streams:
407 maxp_1 = GlobalMaxPooling1D(name='maxp_1')(conv_1)
408 conv_2 = Conv1D(n_conv, k_conv_2, activation='relu',
     ↪  name='conv_2')(drop_embed_layer)
409 maxp_2 = GlobalMaxPooling1D(name='maxp_2')(conv_2)
410 conv_3 = Conv1D(n_conv, k_conv_3,activation='relu',
     ↪  name='conv_3')(drop_embed_layer)
411 maxp_3 = GlobalMaxPooling1D(name='maxp_3')(conv_3)
412 conv_4 = Conv1D(n_conv, k_conv_4, activation='relu',
     ↪  name='conv_4')(drop_embed_layer)
413 maxp_4 = GlobalMaxPooling1D(name='maxp_4')(conv_4)
414 conv_5 = Conv1D(n_conv, k_conv_5, activation='relu',
     ↪  name='conv_5')(drop_embed_layer)
415 maxp_5 = GlobalMaxPooling1D(name='maxp_5')(conv_5)
416 conv_6 = Conv1D(n_conv, k_conv_6, activation='relu',
     ↪  name='conv_6')(drop_embed_layer)
417 maxp_6 = GlobalMaxPooling1D(name='maxp_6')(conv_6)
418 concat12 = concatenate([maxp_1, maxp_2])
419 concat34 = concatenate([maxp_3, maxp_4])
420 concat56 = concatenate([maxp_5, maxp_6])
421 concat = concatenate([concat12, concat34, concat56])
422 dense_layer = Dense(n_dense, activation='relu',
     ↪  name='dense')(concat)          # dense hidden layers:
423 drop_dense_layer = Dropout(dropout,
     ↪  name='drop_dense')(dense_layer)
424 dense_2 = Dense(int(n_dense/4), activation='relu',
     ↪  name='dense_2')(drop_dense_layer)
425 dropout_2 = Dropout(dropout, name='drop_dense_2')(dense_2)
426 predictions = Dense(1, activation='sigmoid',
     ↪  name='output')(dropout_2)          # sigmoid output layer:
427 model5 = Model(input_layer, predictions)          # create model
```

```
428
429 model5.summary()
430
431 plot_model(model5, to_file="m5.png", show_shapes=True,
    ↪ show_layer_names=True)
432
433 model5.compile(loss='binary_crossentropy', optimizer='adam',
    ↪ metrics=['accuracy'])
434 output_dir5 = "model_output/multiconv5" + msglength + version +
    ↪ lang
435 modelcheckpoint = ModelCheckpoint(filepath = output_dir5
    ↪ +"/weights.{epoch:02d}.hdf5")
436 if not os.path.exists(output_dir5):
437     os.makedirs(output_dir5)
438 history5 = model5.fit(training_padded, train_labels,
    ↪ batch_size=batch_size, epochs=epochs,
    ↪ validation_data=(testing_padded, test_labels), callbacks =
    ↪ (modelcheckpoint, early_stop), verbose=1)
439
440 metrics = pd.DataFrame(history5.history)        # Create a
    ↪ dataframe
441 metrics.rename(columns = {'loss': 'Training_Loss', 'accuracy':
    ↪ 'Training_Accuracy','val_loss': 'Validation_Loss',
    ↪ 'val_accuracy': 'Validation_Accuracy'}, inplace = True)
442 def plot_graphs5(var1, var2, string):
443     metrics[[var1, var2]].plot()
444     plt.title('Model: Training and Validation ' + string)
445     plt.xlabel ('Number of epochs')
446     plt.ylabel(string)
447     plt.legend([var1, var2])
448 plot_graphs5('Training_Loss', 'Validation_Loss', 'loss')
449 plot_graphs5('Training_Accuracy', 'Validation_Accuracy',
    ↪ 'accuracy')
450
451 # Comparing different models values
452 print(f"Dense architecture loss and accuracy:
    ↪ {model.evaluate(testing_padded, test_labels)} " )
453 print("**************************************")
454 print(f"LSTM architecture loss and accuracy:
    ↪ {model1.evaluate(testing_padded, test_labels)} " )
```

```
455 print("**************************************")
456 print(f"Bi-LSTM architecture loss and accuracy:
    ↪  {model2.evaluate(testing_padded, test_labels)} " )
457 print("**************************************")
458 print(f"Convolutional architecture loss and accuracy:
    ↪  {model3.evaluate(testing_padded, test_labels)} " )
459 print("**************************************")
460 print(f"Multi-ConvNet architecture loss and accuracy:
    ↪  {model4.evaluate(testing_padded, test_labels)} " )
461 print("**************************************")
462 print(f"Multi-ConvNet 5 architecture loss and accuracy:
    ↪  {model5.evaluate(testing_padded, test_labels)} " )
463
464 # Comparing different models Graph
465 model.load_weights(output_dir + "/weights.03.hdf5")
466 y_hat = model.predict(testing_padded)
467 plt.hist(y_hat)
468 _ = plt.axvline(x=0.5, color='orange')
469
470 "{:0.2f}".format(roc_auc_score(test_labels, y_hat)*100.0)
471 y_hat = np.array([float(i) for i in y_hat])
472 y_hat = np.array([round(i) for i in y_hat])
473 accuracy = accuracy_score(test_labels, y_hat)
474 "{:0.2f}".format((accuracy)*100.0)
475
476 model1.load_weights(output_dir1 + "/weights.03.hdf5")
477 y_hat = model1.predict(testing_padded)
478 plt.hist(y_hat)
479 _ = plt.axvline(x=0.5, color='orange')
480
481 "{:0.2f}".format(roc_auc_score(test_labels, y_hat)*100.0)
482 y_hat = np.array([float(i) for i in y_hat])
483 y_hat = np.array([round(i) for i in y_hat])
484 accuracy = accuracy_score(test_labels, y_hat)
485 "{:0.2f}".format((accuracy)*100.0)
486
487
488 model2.load_weights(output_dir2 + "/weights.03.hdf5")
489 y_hat = model2.predict(testing_padded)
```

```
490 plt.hist(y_hat)
491 _ = plt.axvline(x=0.5, color='orange')
492 
493 "{:0.2f}".format(roc_auc_score(test_labels, y_hat)*100.0)
494 y_hat = np.array([float(i) for i in y_hat])
495 y_hat = np.array([round(i) for i in y_hat])
496 accuracy = accuracy_score(test_labels, y_hat)
497 "{:0.2f}".format((accuracy)*100.0)
498 
499 model3.load_weights(output_dir3 + "/weights.03.hdf5")
500 y_hat = model3.predict(testing_padded)
501 plt.hist(y_hat)
502 _ = plt.axvline(x=0.5, color='orange')
503 
504 "{:0.2f}".format(roc_auc_score(test_labels, y_hat)*100.0)
505 y_hat = np.array([float(i) for i in y_hat])
506 y_hat = np.array([round(i) for i in y_hat])
507 accuracy = accuracy_score(test_labels, y_hat)
508 "{:0.2f}".format((accuracy)*100.0)
509 
510 model4.load_weights(output_dir4 + "/weights.03.hdf5")
511 y_hat = model4.predict(testing_padded)
512 plt.hist(y_hat)
513 _ = plt.axvline(x=0.5, color='orange')
514 
515 "{:0.2f}".format(roc_auc_score(test_labels, y_hat)*100.0)
516 y_hat = np.array([float(i) for i in y_hat])
517 y_hat = np.array([round(i) for i in y_hat])
518 accuracy = accuracy_score(test_labels, y_hat)
519 "{:0.2f}".format((accuracy)*100.0)
520 
521 model.load_weights(output_dir + "/weights.03.hdf5")
522 y_hat = model.predict(testing_padded)
523 plt.hist(y_hat)
524 _ = plt.axvline(x=0.5, color='orange')
525 
526 "{:0.2f}".format(roc_auc_score(test_labels, y_hat)*100.0)
527 y_hat = np.array([float(i) for i in y_hat])
528 y_hat = np.array([round(i) for i in y_hat])
```

```
529 accuracy = accuracy_score(test_labels, y_hat)
530 "{:0.2f}".format((accuracy)*100.0)
531
532 model5.load_weights(output_dir5 + "/weights.03.hdf5")
533 y_hat = model5.predict(testing_padded)
534 plt.hist(y_hat)
535 _ = plt.axvline(x=0.5, color='orange')
536
537 "{:0.2f}".format(roc_auc_score(test_labels, y_hat)*100.0)
538 y_hat = np.array([float(i) for i in y_hat])
539 y_hat = np.array([round(i) for i in y_hat])
540 accuracy = accuracy_score(test_labels, y_hat)
541 "{:0.2f}".format((accuracy)*100.0)
```

Listing 14: antispam.ipynb Es werden die NN trainiert. Die Grundlage für das Traning und Testen der klassischen NN ist dieses Paper: [56].

A.6.2. compare.ipynb

```
1 import numpy as np
2 import pandas as pd
3 import seaborn as sns
4 import matplotlib.pyplot as plt
5 from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
6 %matplotlib inline
7 from sklearn.model_selection import train_test_split        #
   ↪ library for train test split
8 from sklearn.metrics import roc_auc_score          # for
   ↪ roc_auc_score
9 # deep learning libraries for text pre-processing
10 import tensorflow as tf
11 from tensorflow.keras.preprocessing.text import Tokenizer
12 from tensorflow.keras.preprocessing.sequence import pad_sequences
13 from tensorflow.keras.callbacks import EarlyStopping
```

```
14 from tensorflow.keras.models import Sequential
15 from tensorflow.keras.layers import Embedding,
   ↪ GlobalAveragePooling1D,  Flatten, Dense, Dropout, Embedding,
   ↪ LSTM, SpatialDropout1D, Bidirectional, SpatialDropout1D,
   ↪ Conv1D, GlobalMaxPooling1D
16 # Modeling
17 from keras.callbacks import ModelCheckpoint
18 from keras.layers import Input, concatenate # Multi-ConvNet
19 from keras.models import Model
20 from tensorflow import keras
21 import os         # Reading and writing from Disk
22 from tensorflow.keras.utils import plot_model # For Graphic of
   ↪ the NN
23 import pickle    # Load Save Tokenizer
24 
25 print("Num GPUs Available: ",
   ↪ len(tf.config.list_physical_devices('GPU')))
26 
27 version = "66" # nr saved model
28 filename = "lab6_lab_1Monat18042024.csv"
29 allLength = ["short","middle","long","msg"] # fuer vergleich mit
   ↪ Einheitsmodell
30 langList = ["__label__de","__label__en","__label__somtehingElse"]
31 #langList = ["__label__de"]
32 #langList = ["__label__en"]
33 #langList = ["__label__somtehingElse"]
34 #langList = ["all"]
35 lang = "all"
36 # Filter Messages
37 langFilter = "all"
38 #langFilter = "__label__de"
39 #langFilter = "__label__en"
40 #langFilter = "__label__somtehingElse"
41 
42 MAX_LEN = 512 # 300 # pad_sequencs parameter, only look for X
   ↪ words in a sentence
43 VOCAB_SIZE = 32768  #15626   # Dictionary need a fix size
44 trunc_type = "post" # pad_sequences parameter
45 padding_type = "post" # pad_sequences parameter
```

```python
oov_tok = "<OOV>" # out of vocabulary token

compare = [] # values for comparison
dist = [] # y_hat of al versions

def loadAndFilterFile (filename):
    # messages = pd.read_csv(filename, sep =',',names=["label", "message","lang","len"], header = 0) # lab
    messages = pd.read_csv(filename, sep =',',names=["unbekannt","nr","label","message","lang","len","prop"," header = 1) # lab6
    messages.shape
    messages = messages.drop_duplicates()
    messages.shape
    messages.reset_index(drop=True, inplace=True)
    dropList = []
    if(msglength != "msg"):
        cnt = 0
        for msg in messages['len']:
            if(msg != msglength):
                dropList.append(cnt)
            cnt = cnt + 1
        messages.drop(messages.index[dropList], inplace=True) # deletes not used messages
    dropList = []
    cnt = 0
    if(langFilter != "all"):
            for msg in messages['lang']:
                if(msg != langFilter):
                    dropList.append(cnt)
                cnt = cnt + 1
        print(cnt)
        messages.drop(messages.index[dropList], inplace=True) # deletes not used messages
        messages = messages.reset_index()
    #messages.describe()
    return messages

def predict_spam(mod, predict_msg, tokenizer): # takes a spam message an catllculates the probabillity for spam
```

```
80      EMBEDING_DIM = 32  # 16 bei SMS
81      DROP_VALUE = drop_embed = dropout = 0.2 # dropout
82      N_DENSE = 24    #
83      epochs = 30  # default: 4
84      batch_size = 128
85      N_LTSM = 20
86      pad_type = trunc_type = 'pre'   # vector-space embedding
87
88      optim = keras.optimizers.Adam(learning_rate=0.00001) # Using
         ↪ Adam optimiser which makes use of momentum to avoid local
         ↪ minima
89      modelDir = "modelRun/" + mod + msglength + version + lang
90
91      if(mod == "dense0"): #Dense sentiment model architecture
92          model = Sequential()
93          model.add(Embedding(VOCAB_SIZE, EMBEDING_DIM,
             ↪ input_length=MAX_LEN))
94          model.add(GlobalAveragePooling1D())
95          model.add(Dense(N_DENSE, activation='relu'))
96          model.add(Dropout(DROP_VALUE))
97          model.add(Dense(1, activation='sigmoid'))
98      elif(mod == "ltsm"): #LSTM Spam detection architecture
99          #modelDir = "modelRun/ltsm" + msglength + version + lang
100         model = Sequential()
101         model.add(Embedding(VOCAB_SIZE, EMBEDING_DIM,
             ↪ input_length=MAX_LEN))
102         model.add(LSTM(N_LTSM, dropout=DROP_VALUE,
             ↪ return_sequences=True))
103         model.add(LSTM(N_LTSM, dropout=DROP_VALUE,
             ↪ return_sequences=True))
104         model.add(GlobalAveragePooling1D())
105         model.add(Dense(1, activation='sigmoid'))
106     elif(mod == "bi-ltsm"):# Biderectional LSTM Spam detection
         ↪ architecture
107         model = Sequential()
108         model.add(Embedding(VOCAB_SIZE, EMBEDING_DIM,
             ↪ input_length=MAX_LEN))
109         model.add(Bidirectional(LSTM(N_LTSM, dropout=DROP_VALUE,
             ↪ return_sequences=True)))
```

```
110            model.add(GlobalAveragePooling1D())
111            model.add(Dense(1, activation='sigmoid'))
112        elif(mod == "conv3"):       # deep-learning-illustrated-
            ↪  master/notebooks/convolutional_sentiment_classifier.ipynb
113            max_review_length = 400
114            N_CONV = 256 # filters, a.k.a. kernels
115            k_conv = 3 # kernel length
116            N_DENSE = 256
117            model = Sequential()
118            model.add(Embedding(VOCAB_SIZE, EMBEDING_DIM,
                ↪  input_length=MAX_LEN))
119            model.add(SpatialDropout1D(DROP_VALUE))
120            model.add(Conv1D(N_CONV, k_conv, activation='relu'))
121            model.add(Conv1D(N_CONV, k_conv, activation='relu'))
122            model.add(GlobalMaxPooling1D())
123            model.add(Dense(N_DENSE, activation='relu'))
124            model.add(Dropout(DROP_VALUE))
125            model.add(Dense(1, activation='sigmoid'))
126        elif(mod == "multiconv"):
127            n_conv_1 = n_conv_2 = n_conv_3 = 256    # convolutional
                ↪  layer architecture
128            k_conv_1 = 3
129            k_conv_2 = 2
130            k_conv_3 = 4
131            n_dense = 256   # dense layer architecture
132            input_layer = Input(shape=(MAX_LEN,), dtype='int16',
                ↪  name='input')
133            embedding_layer = Embedding(VOCAB_SIZE, EMBEDING_DIM,
                ↪  name='embedding')(input_layer)     # embedding
134            drop_embed_layer = SpatialDropout1D(drop_embed,
                ↪  name='drop_embed')(embedding_layer)
135            conv_1 = Conv1D(n_conv_1, k_conv_1, activation='relu',
                ↪  name='conv_1')(drop_embed_layer) # three parallel
                ↪  convolutional streams:
136            maxp_1 = GlobalMaxPooling1D(name='maxp_1')(conv_1)
137            conv_2 = Conv1D(n_conv_2, k_conv_2, activation='relu',
                ↪  name='conv_2')(drop_embed_layer)
138            maxp_2 = GlobalMaxPooling1D(name='maxp_2')(conv_2)
139            conv_3 = Conv1D(n_conv_3, k_conv_3, activation='relu',
                ↪  name='conv_3')(drop_embed_layer)
```

```
140         maxp_3 = GlobalMaxPooling1D(name='maxp_3')(conv_3)
141         concat = concatenate([maxp_1, maxp_2, maxp_3])   #
             ↪  concatenate the activations from the three streams
142         dense_layer = Dense(n_dense, activation='relu',
             ↪  name='dense')(concat) # dense hidden layers
143         drop_dense_layer = Dropout(dropout,
             ↪  name='drop_dense')(dense_layer)
144         dense_2 = Dense(int(n_dense/4), activation='relu',
             ↪  name='dense_2')(drop_dense_layer)
145         dropout_2 = Dropout(dropout,
             ↪  name='drop_dense_2')(dense_2)
146         predictions = Dense(1, activation='sigmoid',
             ↪  name='output')(dropout_2) # sigmoid output layer
147         model = Model(input_layer, predictions) # create model
148     elif(mod == "multiconv5"):
149         n_conv = 768    # convolutional layer architecture:
150         k_conv_1 = 1
151         k_conv_2 = 2
152         k_conv_3 = 3
153         k_conv_4 = 5
154         k_conv_5 = 7
155         k_conv_6 = 9
156         n_dense = 512   # dense layer architecture:
157         input_layer = Input(shape=(MAX_LEN,), dtype='int16',
             ↪  name='input')
158         embedding_layer = Embedding(VOCAB_SIZE, EMBEDING_DIM,
             ↪  name='embedding')(input_layer)    # embedding:
159         drop_embed_layer = SpatialDropout1D(drop_embed,
             ↪  name='drop_embed')(embedding_layer)
160         conv_1 = Conv1D(n_conv, k_conv_1, activation='relu',
             ↪  name='conv_1')(drop_embed_layer)   # three parallel
             ↪  convolutional streams:
161         maxp_1 = GlobalMaxPooling1D(name='maxp_1')(conv_1)
162         conv_2 = Conv1D(n_conv, k_conv_2, activation='relu',
             ↪  name='conv_2')(drop_embed_layer)
163         maxp_2 = GlobalMaxPooling1D(name='maxp_2')(conv_2)
164         conv_3 = Conv1D(n_conv, k_conv_3, activation='relu',
             ↪  name='conv_3')(drop_embed_layer)
165         maxp_3 = GlobalMaxPooling1D(name='maxp_3')(conv_3)
```

```
166         conv_4 = Conv1D(n_conv, k_conv_4, activation='relu',
             ↪ name='conv_4')(drop_embed_layer)
167         maxp_4 = GlobalMaxPooling1D(name='maxp_4')(conv_4)
168         conv_5 = Conv1D(n_conv, k_conv_5, activation='relu',
             ↪ name='conv_5')(drop_embed_layer)
169         maxp_5 = GlobalMaxPooling1D(name='maxp_5')(conv_5)
170         conv_6 = Conv1D(n_conv, k_conv_6, activation='relu',
             ↪ name='conv_6')(drop_embed_layer)
171         maxp_6 = GlobalMaxPooling1D(name='maxp_6')(conv_6)
172         concat12 = concatenate([maxp_1, maxp_2])
173         concat34 = concatenate([maxp_3, maxp_4])
174         concat56 = concatenate([maxp_5, maxp_6])
175         concat = concatenate([concat12, concat34, concat56])
176         dense_layer = Dense(n_dense, activation='relu',
             ↪ name='dense')(concat)    # dense hidden layers:
177         drop_dense_layer = Dropout(dropout,
             ↪ name='drop_dense')(dense_layer)
178         dense_2 = Dense(int(n_dense/4), activation='relu',
             ↪ name='dense_2')(drop_dense_layer)
179         dropout_2 = Dropout(dropout,
             ↪ name='drop_dense_2')(dense_2)
180         predictions = Dense(1, activation='sigmoid',
             ↪ name='output')(dropout_2)  # sigmoid output layer:
181         model = Model(input_layer, predictions) # create model
182 
183     model.load_weights(modelDir + ".hdf5")
184     new_seq = tokenizer.texts_to_sequences(predict_msg)
185     padded = pad_sequences(new_seq, maxlen =MAX_LEN, padding =
         ↪ padding_type, truncating=trunc_type)
186     return (model.predict(padded))# returns the probabillity for
         ↪ spam
187 
188 def count_results(mod, msglength, version, lang, y_hat,
     ↪ messages): # all global variables
189     ham = 0
190     spam = 0
191     unsure = 0
192     falsepositiv = 0
193     falsenegativ = 0
```

```
194     cnt = 0
195     for i in y_hat:
196         if (i < .5):
197             #print(str(cnt) + " " + messages.loc[cnt,'label'])
198             if(messages.loc[cnt,'label'] == 'ham'):
199                     ham = ham + 1
200             else:
201                     falsenegativ = falsenegativ + 1
202         elif (i > .5):
203             if(messages.loc[cnt,'label'] == 'spam'):
204                     spam = spam +1
205             else:
206                     falsepositiv = falsepositiv + 1
207         cnt = cnt + 1
208     compare.append([mod, msglength, version, lang, ham,
         ↪ falsenegativ, unsure, falsepositiv, spam, ham+spam,
         ↪ falsenegativ+unsure+falsepositiv, cnt])
209
210 def loadAndComputeModelResults(msglength, version, lang,
     ↪ filename):
211     messages = loadAndFilterFile(filename)
212     tokenDir= "model_output/" + msglength + version + lang + "/"
213     with open(tokenDir+ 'tokenizer.pickle', 'rb') as handle:
214         tokenizer = pickle.load(handle)
215     modList = ["dense0", "ltsm", "bi-ltsm", "conv3", "multiconv",
         ↪ "multiconv5"]
216     for mod in modList:
217         y_hat =(predict_spam(mod, messages['message'],
             ↪ tokenizer)) # predict SPAM
218         dist.append([mod, msglength, version, lang, y_hat]) #
             ↪ Save Results
219         count_results(mod, msglength, version, lang, y_hat,
             ↪ messages)
220
221 for msglength in allLength:
222     for lang in langList:
223         loadAndComputeModelResults(msglength, version, lang,
             ↪ filename)
224
```

```
225 print(" Version:" + str(compare[0][2]))
226 print("Modell\tlen\tlang\tham\tfalseN\tfalseP\tspam\tright\twrong")
227 for line in compare:
228     if (line[0] == "dense0"):
229         print(" Mails:"+str(line[11])
            ↪  +"\t----------------------------------")
230     print(str(line[0][0:6])+"\t"+str(line[1])+i
        ↪  "\t"+str(line[3][9:11])+"\t"+str(line[4])+"\t"+str(line[5])+"\t"+
        ↪  str(line[7])+"\t"+str(line[8])+"\t"
        ↪  +str(line[9])+"\t"+str(line[10]))
231 
232 print("Modell\tlen\tvers\tlang\twrong/cnt in %i\n")
233 bestModell = "dens0"
234 bestOptions = ""
235 bestValue = 100.0
236 bestList = []
237 for line in compare:
238     if (line[0] == "dense0"):
239         print("----------------------------------")
240     if(line[9] < 1):# if devision 0
241         error = 9999
242     else:
243         error = (100*(line[10])/int(line[11]))
244     if(error < bestValue):
245         bestModell = line[0]
246         bestOptions =
            ↪  str(line[1]),str(line[2]),str(line[3][9:11])
247         wrongCnt  = line[10]
248         mailCnt   = line[11]
249         bestValue = error
250         hamCnt    = line[5]
251         falseNcnt = line[6]
252         falsePcnt = line[7]
253         spamCnt   = line[8]
254         rightCnt  = line[9]
255     if(line[9] < 1):# if devision 0
256         print(str(line[0][0:6]) + "\t" +str(line[1]) + "\t"
            ↪  +str(line[2]) + "\t" + str(line[3][9:11]) + "\t" +
            ↪  "all False!")
```

```
257        else:
258            print(str(line[0][0:6]) + "\t" +str(line[1]) + "\t"
                ↪  +str(line[2]) + "\t" + str(line[3][9:11]) + "\t" +
                ↪  str(100*int(line[10])/int(line[11])))
259        if (line[0] == "multiconv5"):
260
                ↪  print(bestModell+"\t"+str(bestOptions[0])+str(bestOptions[1])+str(
261            bestList.append([bestOptions, bestModell, bestValue,
                ↪  wrongCnt, mailCnt, hamCnt, falseNcnt, falsePcnt,
                ↪  spamCnt, rightCnt])
262            bestModell = "dens0"
263            bestValue = 100.0
264
265    sumList = []
266    for best in bestList:
267        print(best[0][0]+"\t"+best[0][1]+"\t"+best[0][2]+"\t"+
            ↪  str(best[2])+"\t"+str(best[1]))
268        sumList.append([best[3],best[4]])
269        print(best)
270    wrongCnt = 0
271    mailCnt = 0
272    print("\n")
273    for data in sumList:
274        print(str(data[0])+"\t"+str(data[1])+"\t"+str(100*(
            ↪  data[0]/data[1])))
275        wrongCnt = wrongCnt + data[0]
276        mailCnt = mailCnt + data[1]
277    print("\n")
278    print("wrongC\tmailC\t%")
279    print(str(wrongCnt)+"\t"+str(mailCnt)+"\t"+str(100*(
        ↪  wrongCnt/mailCnt)))
280
281    for data in dist:
282        #plt.subplot(3,3,1)
283        plt.hist(data[4]) # 4. is distribution
284        _ = plt.axvline(x=0.5, color='orange')
285        plt.text(1.1, 1, str(data[0] + " " + data[1] + " " + data[2]
            ↪  + " " + data[3]))
286        plt.show()
```

```
287
288 with open(filename + '.pickle', 'wb') as handle:        # Save
    ↪  best Model List in File
289     pickle.dump(bestList, handle,
        ↪  protocol=pickle.HIGHEST_PROTOCOL)
```

Listing 15: compare.ipynb Vergleich mehre NN miteinander.

A.7. Training und Tests LLM

A.7.1. llm-classifier.ipynb

```
1  get_ipython().system('nvcc -version')
2
3  import numpy as np
4  import pandas as pd
5
6  from transformers import (
7      AutoConfig,
8      AutoTokenizer,
9      AutoModelForCausalLM,
10     get_linear_schedule_with_warmup,
11     get_cosine_schedule_with_warmup
12 )
13
14 import os
15 import time
16 import zipfile
17 import urllib.request
18 from pathlib import Path
19 from tqdm.auto import tqdm
20
21 import torch
22 from torch import nn, optim
```

```
23 import torch.nn.functional as F
24 from torch.amp import GradScaler, autocast
25 from torch.utils.data import Dataset, DataLoader
26
27 from sklearn.model_selection import StratifiedKFold
28 from sklearn.metrics import log_loss, accuracy_score
29 import lightning as L
30
31 tqdm.pandas()
32 os.environ['TOKENIZERS_PARALLELISM'] = 'false'
33
34 model_id = 'phi-4'
35
36 filename = "6jSpamHam1q24_v46.csv"
37
38 df = pd.read_csv(filename, sep
    ↪ =',',names=["unnamed","nr","label","text","lang","prop","len","nil"],
    ↪ header = 1) # mit Kopfzeile
39
40 df.shape
41
42 num_epochs = 1 # default ist 1
43
44 df=df.drop_duplicates()
45 df = df[~df.index.duplicated(keep='first')]
46 df
47 df.shape
48
49 df = df.dropna() # loescht leere gibt sonst problme beim
    ↪ Trainning
50 df.shape
51
52 def create_balanced_dataset(df):
53     num_spam = df[df["label"] == "spam"].shape[0] # Count the
        ↪ instances of "spam"
54     ham_subset = df[df["label"] == "ham"].sample(num_spam,
        ↪ random_state=123)         # Randomly sample "ham"
        ↪ instances to match the number of "spam" instances
55     balanced_df = pd.concat([ham_subset, df[df["label"] ==
        ↪ "spam"]])     # Combine ham "subset" with "spam"
```

```
56      return balanced_df
57
58  balanced_df = create_balanced_dataset(df)
59  print(balanced_df["label"].value_counts())
60
61  balanced_df['label'] = df.label.map({'spam': 1, 'ham': 0})
62
63  def random_split(df, train_frac, validation_frac):
64      df = df.sample(frac=1,
        ↪  random_state=123).reset_index(drop=True)          # Shuffle
        ↪  the entire DataFrame
65      train_end = int(len(df) * train_frac)          # Calculate
        ↪  split indices
66      validation_end = train_end + int(len(df) * validation_frac)
67      # Split the DataFrame
68      train_df = df[:train_end]
69      validation_df = df[train_end:validation_end]
70      test_df = df[validation_end:]
71
72      return train_df, validation_df, test_df
73
74  train_df, validation_df, test_df = random_split(balanced_df, 0.7,
    ↪  0.1)          # Test size is implied to be 0.2 as the remainder
75
76  train_df.to_csv("train.csv", index=None)
77  validation_df.to_csv("validation.csv", index=None)
78  test_df.to_csv("test.csv", index=None)
79
80  get_ipython().system('ls')
81
82  # # Load tokenizer
83  tokenizer =
    ↪  AutoTokenizer.from_pretrained(model_id,local_files_only=True)
84
85  # Set padding token to end-of-sequence token and configure
    ↪  padding side
86  tokenizer.pad_token = tokenizer.eos_token
87  tokenizer.padding_side = 'left'
88
```

```
# # Split data
train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')
val = pd.read_csv('validation.csv')

train['text'] = tokenizer.bos_token + train['text']
test['text'] = tokenizer.bos_token + test['text']
val['text'] = tokenizer.bos_token + val['text']

sample = tokenizer(train.text[0],
    ↪ add_special_tokens=False).input_ids
tokenizer.decode(sample)

class CustomDataset(Dataset):
    def __init__(self, texts, targets):
        self.texts = texts
        self.targets = torch.tensor(targets, dtype=torch.long)

    def __getitem__(self, idx):
        text = self.texts[idx]
        target = self.targets[idx]
        return text, target

    def __len__(self):
        return len(self.targets)

# Set seed for reproducibility
L.seed_everything(seed=252)

# Create train dataset and dataloader
train_dataset = CustomDataset(
    texts=train['text'].values.tolist(),
    targets=train['label'].values.tolist()
)
train_dataloader = DataLoader(
    dataset=train_dataset,
    batch_size=8,
    shuffle=True,
    drop_last=True
```

```
127 )
128
129 # Create test dataset and dataloader
130 test_dataset = CustomDataset(
131     texts=test['text'].values.tolist(),
132     targets=test['label'].values.tolist()
133 )
134 test_dataloader = DataLoader(
135     dataset=test_dataset,
136     batch_size=16,
137     shuffle=False,
138     drop_last=False
139 )
140
141 # Create validation dataset and dataloader
142 val_dataset = CustomDataset(
143     texts=val['text'].values.tolist(),
144     targets=val['label'].values.tolist()
145 )
146 val_dataloader = DataLoader(
147     dataset=val_dataset,
148     batch_size=16,
149     shuffle=False,
150     drop_last=False
151 )
152
153 # # Tokenization function
154 def tokenize_text(text):         # Tokenize the text and return
     ↪  PyTorch tensors with dynamic padding
155     encodings = tokenizer(
156         text,
157         return_tensors='pt',
158         padding='max_length',  # max length taken from Model
             ↪  tokenizer_config.json model_max_length, there fixed
             ↪  to 512. For compare with other Modells
159         truncation=True,
160         add_special_tokens=False
161     )
162
```

```
163         return encodings
164 
165     # # Architecture
166     torch.cuda.device_count()
167 
168     torch.cuda.current_device()
169 
170     cuda1 = torch.cuda.set_device(1)         # use second GPU
171 
172     torch.cuda.current_device()
173 
174     device = 'cuda' if torch.cuda.is_available() else 'cpu'
175     print(f'Using device: {device}')
176 
177     def disable_dropout(model: torch.nn.Module):         #Disable
          ↪ dropout in a model.
178         for module in model.modules():
179             if isinstance(module, torch.nn.Dropout):
180                 module.p = 0
181 
182     # Define a neural network class
183     class Net(nn.Module):
184         def __init__(self):
185             super(Net, self).__init__()
186 
187             # Load configuration from a pre-trained model
188             config = AutoConfig.from_pretrained(model_id)
189 
190             # Load pre-trained language model with specific
              ↪ configurations
191             self.llm = AutoModelForCausalLM.from_pretrained(
192                 model_id,
193                 device_map=device,
194                 low_cpu_mem_usage=True,
195                 torch_dtype=torch.float16,
196                 trust_remote_code=True,
197             )
198 
199             # Replace language model head with an identity function
```

```
200            self.llm.lm_head = nn.Identity()
201
202            # Freeze all parameters of the language model backbone
203            for name, param in self.llm.named_parameters():
204                param.requires_grad = False
205
206            # Define classification head
207            self.cls_head = nn.Sequential(
208                nn.Linear(config.hidden_size, 768),
209                nn.ReLU(),
210                nn.LayerNorm(768),
211                nn.Linear(768, 2)
212            )
213
214        # Define the forward pass
215        def forward(self, input_ids, attention_mask):
216            x = self.llm(input_ids, attention_mask).logits  # get
                ↪  last hidden state
217            logits = self.cls_head(x)[:, -1, :]  # Apply
                ↪  classification head to the last token's output
218            return logits
219
220
221    # # Optimizer and Scheduler
222    def get_optimizer(model, learning_rate=0.0001, diff_lr=0.00001,
        ↪  weight_decay=0.01):
223        """
224        Get optimizer with different learning rates for specified
        ↪  layers.
225
226        Args:
227            model (torch.nn.Module): The neural network model.
228            learning_rate (float): Learning rate for non-differential
        ↪  layers.
229            diff_lr (float): Learning rate for differential layers.
230            weight_decay (float): Weight decay (decoupled from L2
        ↪  penalty) for optimizer.
231
232        Returns:
```

```
233             torch.optim.AdamW: Optimizer for the model.
234         """
235         # Define parameters with different learning rates and weight
            ↪  decay
236         no_decay = ['bias', 'LayerNorm.weight']
237         differential_layers = ['llm']
238 
239         optimizer = torch.optim.AdamW(
240                 [
241                     {
242                         "params": [
243                             param
244                             for name, param in
                                 ↪  model.named_parameters()
245                             if (not any(layer in name for layer in
                                 ↪  differential_layers))
246                             and (not any(nd in name for nd in
                                 ↪  no_decay))
247                         ],
248                         "lr": learning_rate,
249                         "weight_decay": weight_decay,
250                     },
251                     {
252                         "params": [
253                             param
254                             for name, param in
                                 ↪  model.named_parameters()
255                             if (not any(layer in name for layer in
                                 ↪  differential_layers))
256                             and (any(nd in name for nd in no_decay))
257                         ],
258                         "lr": learning_rate,
259                         "weight_decay": 0,
260                     },
261                     {
262                         "params": [
263                             param
264                             for name, param in
                                 ↪  model.named_parameters()
```

```
265                             if (any(layer in name for layer in
                                 ↪  differential_layers))
266                             and (not any(nd in name for nd in
                                 ↪  no_decay))
267                         ],
268                         "lr": diff_lr,
269                         "weight_decay": weight_decay,
270                     },
271                     {
272                         "params": [
273                             param
274                             for name, param in
                                 ↪  model.named_parameters()
275                             if (any(layer in name for layer in
                                 ↪  differential_layers))
276                             and (any(nd in name for nd in no_decay))
277                         ],
278                         "lr": diff_lr,
279                         "weight_decay": 0,
280                     },
281                 ],
282                 lr=learning_rate,
283                 weight_decay=weight_decay,
284         )
285 
286         return optimizer
287 
288 # # Hyperameters
289 learning_rate = 0.0002
290 diff_lr = 0.00001 # not being used because I freeze the llm
     ↪  backbone
291 warmup_steps = 0
292 seed = 252
293 weight_decay = 0.01
294 
295 # # Fine-tuning
296 L.seed_everything(seed=seed)          # Set seed for
     ↪  reproducibility
297 
```

```
298 # Instantiate the neural network model
299 model = Net()
300 model.to(device)  # Move model to the device
301 
302 # Display the names of trainable parameters
303 print('Here are the trainable parameters:')
304 for n, p in model.named_parameters():
305     if p.requires_grad:
306         print(n)
307 
308 # Get the optimizer
309 optimizer = get_optimizer(
310     model,
311     learning_rate=learning_rate,
312     diff_lr=diff_lr,
313     weight_decay=weight_decay
314 )
315 
316 # Set up the scheduler for learning rate adjustment
317 scheduler = get_linear_schedule_with_warmup(
318     optimizer=optimizer,
319     num_warmup_steps=warmup_steps,
320     num_training_steps=num_epochs*len(train_dataloader)
321 )
322 
323 scaler = GradScaler()
324 history = []
325 start_time = time.time()
326 for epoch in range(num_epochs):
327 
328     for batch_idx, batch in enumerate(train_dataloader):
329 
330         model.train()
331 
332         prompt, targets = batch
333 
334         encodings = tokenize_text(prompt)
335 
336         input_ids = encodings['input_ids'].to(device)
```

```
337            attention_mask = encodings['attention_mask'].to(device)
338            targets = targets.to(device)
339
340            # Perform forward pass with autocast for mixed precision
               ↪ training
341            with autocast(device):
342                logits = model(input_ids, attention_mask)
343                loss = F.cross_entropy(logits, targets)
344
345            # Backward pass, optimization step, and learning rate
               ↪ adjustment
346            optimizer.zero_grad()
347            scaler.scale(loss).backward()
348            scaler.step(optimizer)
349            scaler.update()
350            scheduler.step()
351            history.append([epoch, batch_idx, loss.item()])    # save
               ↪ data for graph
352            # Logging training progress
353            if (batch_idx % 100 == 1):
354                print(
355                    f'Epoch: {epoch+1} / {num_epochs}'
356                    f'| Batch: {batch_idx+1}/{len(train_dataloader)}'
357                    f'| Loss: {loss.item():.4f}'
358                )
359
360    end_time = time.time()
361    training_time = (end_time - start_time) / 60
362    print(f'Total training time: {training_time:.2f} min')
363
364
365    # # Evaluation
366    def calc_accuracy(dataloader, type):
367        with torch.no_grad():
368            model.eval()
369            pred_scores = []
370            actual_scores = []
371            for batch in tqdm(dataloader, total=len(dataloader),
               ↪ desc=f'Calc {type} accuracy'):
```

```
372                 prompt, targets = batch
373                 encodings = tokenize_text(prompt)
374 
375                 input_ids = encodings['input_ids'].to(device)
376                 attention_mask =
                     ↪  encodings['attention_mask'].to(device)
377 
378                 with autocast(device):
379                     logits = model(input_ids, attention_mask)
380                     pred_score = F.softmax(logits,
                         ↪  dim=-1).argmax(dim=-1).cpu().detach().numpy().tolist()
381                     pred_scores.extend(pred_score)
382                     actual_scores.extend(targets.numpy().tolist())
383             pred_scores = np.array(pred_scores)
384             accuracy = accuracy_score(actual_scores, pred_scores)
385             return accuracy
386 
387 train_acc = calc_accuracy(train_dataloader, type='train')
388 print('Train accuracy:', train_acc)
389 test_acc = calc_accuracy(test_dataloader, type='test')
390 print('Test accuracy:', test_acc)
391 val_acc = calc_accuracy(val_dataloader, type='val')
392 print('Val accuracy:', val_acc)
393 
394 print('Train accuracy:', train_acc)
395 print('Test accuracy:', test_acc)
396 print('Val accuracy:', val_acc)
```

Listing 16: llm-classifier.ipynb Es werden die LLMs trainiert und getestet. Grundlage für das Jupyter-Notebook für das Training und Testen von LLMs ist von hier: [186].

A.7.2. csv2jsonAlpaca3.ipynb

```
# * Soll
# * * die Mails im CSV-Format einlesen
# * * Text bearbeiten
# * * es ins JSON Format umwandeln und speichern
# * * Ausgabe in alp_Originalname.json
import numpy as np
import pandas as pd
import os # Reading and writing from Disk

get_ipython().system(' pip list')
filename = "mail65plus.csv"  # de, en, it, sk, hu, fr ro nl zh cs
↪ es pt tr
messages = pd.read_csv(filename, sep =',',names=["output",
↪ "input"], header = 0)

type(messages)
messages.info()
messages.shape
messages.size
messages.dtypes
messages.columns
messages=messages.drop_duplicates() # Delete duplicates
messages.describe()

inputList = []
outputList = []
inputList = []
instructionList = []
for index,msg in messages.iterrows():
    instructionList.append("Is this eMail message Spam or Ham?")
    inputList.append(msg['input'])
    outputList.append(msg['output'])

len(instructionList)

len(inputList)

len(outputList)

```

```
38 pdout = pd.DataFrame(instructionList)
39 pdout.columns=["instruction"]
40 pdout["input"] = inputList
41 pdout["output"] = outputList
42
43 pdout.groupby('output').describe()
44
45 outputFilename = "alp_" + filename + ".json"
46 pdout.to_json(outputFilename, orient='records', lines=True)
```

Listing 17: csv2jsonAlpaca3.ipynb Wandelt den Mail-Datensatz in Json um.

A.8. NN Auswahl

A.8.1. cpKiMpodeli.py

```
1  #cpKiMpodel kopiert die KI-Modelle in eine eigenes Verzeichniss
   ↪  mit neuen Namen und zwar nur jeweils das 3. von hinten, nur
   ↪  fuer die angegeben VersionNummer
2  import os
3  from pathlib import Path     # for reding dirs
4  from shutil import copyfile # for copy
5
6  dirpath      = "model_output/"
7  outputDir    = "modelRun/"
8  modelList    = []     # stores all to copy KI Model Files
9  destList     = []     # destionation Name of Model-Files
10
11 paths = sorted(Path(dirpath).iterdir(), key=os.path.getmtime)    #
   ↪  newest file at end of list
12 for data in paths:
13     subpaths = sorted(Path(data).iterdir(), key=os.path.getmtime)
        ↪  # newest file at end of list
14     if(len(subpaths)>= 3):
```

```
15          modelList.append(subpaths[-3])
16  if not os.path.exists(outputDir):
17      os.makedirs(outputDir)
18      print("mkdir "+ outputDir)
19  for subdata in modelList:
20      pathName = outputDir + str(str(subdata).split("/")[1]) +
     ↪   ".hdf5"
21      print("cp " + str(subdata) + " " +pathName)
22      destList.append(pathName)
23      copyfile(subdata,pathName)
```

Listing 18: cpKiModel.py Speichert den besten Snapshot von klassischen NN in eignen Dateien für den Vergleich und die spätere Anwendung.

A.8.2. bestModeledit.ipynb

```
1   import os# Reading and writing from Disk
2   from pathlib import Path     # for reding dirs
3   import pickle    # Load Save Tokenizer
4
5   dirpath      = "."
6   pickleFiles = []
7
8   paths = sorted(Path(dirpath).iterdir(), key=os.path.getmtime)    #
     ↪  newest file at end of list
9   for data in paths:
10      if(".pickle" in str(data)):
11          print(data)
12          pickleFiles.append(data)
13
14  bestAll = []
15  for file in pickleFiles: # load all files
16      print(file)
17      with open(file, 'rb') as handle:
```

```
18          bestList = pickle.load(handle)
19          for best in bestList:
20              print(best)
21              bestAll.append(best)
22
23  cnt = 0
24  for data in bestAll:
25      print(str(cnt)+"\t"+str(data))
26      cnt = cnt + 1
27
28  Hier noch die newList anpassen und dann diese Zeile
     ↪  auskommentieren
29  filename = "ham2021spam4opti1" # Name der neuen Datei
30  newList = [0,1,2,3,4,5,6,7,8,9,10,11,12] # Nummern der neuen
     ↪  Auswahl
31
32  bestNew=[]
33  for cnt in newList:
34      print(str(cnt)+"\t"+str(bestAll[cnt]))
35      bestNew.append(bestAll[cnt])
36
37  with open(filename + '.pickle', 'wb') as handle: # Save best
     ↪  Model List in File
38      pickle.dump(bestNew, handle,
         ↪  protocol=pickle.HIGHEST_PROTOCOL)
```

Listing 19: bestModeledit.ipynb Wählt das beste klassische NN aus.

B. Anhang Python-Pakete

B.1. laptop.pip

```
Package                   Version
---------------- ---------
appdirs                   1.4.4
argon2-cffi               18.3.0
attrs                     20.3.0
backcall                  0.2.0
bcrypt                    3.1.7
beautifulsoup4            4.9.3
bleach                    3.2.1
blinker                   1.4
Brlapi                    0.8.2
certifi                   2020.6.20
chardet                   4.0.0
cryptography              3.3.2
cupshelpers               1.0
Cython                    0.29.21
dbus-python               1.2.16
decorator                 4.4.2
defusedxml                0.6.0
distlib                   0.3.1
distro                    1.5.0
distro-info               1.0
duplicity                 0.8.17
entrypoints               0.3
exceptiongroup            1.2.2
fasteners                 0.14.1
filelock                  3.0.12
future                    0.18.2
```

```
29 h5py                         2.10.0
30 h5py.-debian-h5py-serial 2.10.0
31 html5lib                     1.1
32 httplib2                     0.18.1
33 idna                         2.10
34 importlib-metadata           1.6.0
35 iniconfig                    2.0.0
36 invoke                       1.4.1
37 ipykernel                    5.4.3
38 ipython                      7.20.0
39 ipython-genutils             0.2.0
40 ipywidgets                   6.0.0
41 jedi                         0.18.0
42 Jinja2                       2.11.3
43 jsonschema                   3.2.0
44 jupyter-client               6.1.11
45 jupyter-console              6.2.0
46 jupyter-core                 4.7.1
47 Keras                        2.3.1
48 Keras-Applications           1.0.8
49 Keras-Preprocessing          1.1.0
50 lockfile                     0.12.2
51 louis                        3.16.0
52 lxml                         4.6.3
53 Mako                         1.1.3
54 MarkupSafe                   1.1.1
55 mistune                      0.8.4
56 monotonic                    1.5
57 more-itertools               4.2.0
58 nbconvert                    5.6.1
59 nbformat                     5.1.2
60 nose                         1.3.7
61 notebook                     6.2.0
62 numpy                        1.19.5
63 oauthlib                     3.1.0
64 olefile                      0.46
65 packaging                    20.9
66 pandocfilters                1.4.3
67 parameterized                0.7.0
```

```
68  paramiko                 2.7.2
69  parso                    0.8.1
70  pexpect                  4.8.0
71  pickleshare              0.7.5
72  Pillow                   8.1.2
73  pip                      20.3.4
74  pluggy                   1.5.0
75  prometheus-client        0.9.0
76  prompt-toolkit           3.0.14
77  psutil                   5.8.0
78  pycairo                  1.16.2
79  pycups                   2.0.1
80  pycurl                   7.43.0.6
81  pydot                    1.4.2
82  pydotplus                2.0.2
83  Pygments                 2.7.1
84  PyGObject                3.38.0
85  pygpu                    0.7.6
86  pygraphviz               1.7
87  pyinotify                0.9.6
88  PyJWT                    1.7.1
89  PyNaCl                   1.4.0
90  pyparsing                2.4.7
91  pyrsistent               0.15.5
92  PySimpleSOAP             1.16.2
93  pysmbc                   1.0.23
94  pytest                   8.3.4
95  python-apt               2.2.1
96  python-dateutil          2.8.1
97  python-debian            0.1.39
98  python-debianbts         3.1.0
99  python-pam               1.8.4
100 python-xapp              2.0.2
101 python-xlib              0.29
102 pytz                     2021.1
103 pyxdg                    0.27
104 PyYAML                   5.3.1
105 pyzmq                    20.0.0
106 reportbug                7.10.3+deb11u1
```

```
107 requests                  2.25.1
108 scipy                     1.6.0
109 scour                     0.38.2
110 Send2Trash                1.6.0b1
111 setproctitle              1.2.1
112 setuptools                52.0.0
113 six                       1.16.0
114 soupsieve                 2.2.1
115 terminado                 0.9.2
116 testpath                  0.4.4
117 Theano                    1.0.5
118 tinycss                   0.4
119 tinycss2                  1.0.2
120 tomli                     2.2.1
121 tornado                   6.1
122 tqdm                      4.67.1
123 traitlets                 5.0.5
124 unattended-upgrades       0.1
125 urllib3                   1.26.5
126 virtualenv                20.4.0+ds
127 visidata                  2.2.1
128 wcwidth                   0.1.9
129 webencodings              0.5.1
130 wheel                     0.34.2
131 widgetsnbextension        2.0.0
132 xdg                       5
133 zipp                      1.0.0
```

Listing 20: laptop.pip

B.2. notebook.pip

```
absl-py                       2.1.0
annotated-types               0.7.0
appdirs                       1.4.4
argon2-cffi                   18.3.0
argostranslate                1.9.6
astunparse                    1.6.3
attrs                         20.3.0
backcall                      0.2.0
bcrypt                        3.1.7
beautifulsoup4                4.9.3
bleach                        3.2.1
blinker                       1.4
blis                          1.0.1
cachetools                    4.2.4
catalogue                     2.0.10
certifi                       2020.6.20
cffi                          1.17.1
chardet                       4.0.0
clang                         5.0
click                         8.1.7
cloudpathlib                  0.20.0
colorama                      0.4.4
confection                    0.1.5
contourpy                     1.3.0
cryptography                  3.3.2
ctranslate2                   4.5.0
cupshelpers                   1.0
cycler                        0.12.1
cymem                         2.0.10
Cython                        3.0.11
dbus-python                   1.2.16
decorator                     4.4.2
defusedxml                    0.6.0
distlib                       0.3.9
distro                        1.5.0
distro-info                   1.0+deb11u1
entrypoints                   0.3
fasteners                     0.14.1
fasttext                      0.9.3
```

```
40 filelock                      3.16.1
41 flatbuffers                   24.3.25
42 fonttools                     4.55.2
43 fsspec                        2024.10.0
44 future                        0.18.2
45 gast                          0.4.0
46 google-auth                   1.35.0
47 google-auth-oauthlib          0.4.6
48 google-pasta                  0.2.0
49 grpcio                        1.68.1
50 h5py                          3.12.1
51 h5py.-debian-h5py-serial      2.10.0
52 html5lib                      1.1
53 httplib2                      0.18.1
54 idna                          2.10
55 importlib-metadata            8.5.0
56 importlib-resources           6.4.5
57 invoke                        1.4.1
58 ipaddress                     1.0.23
59 ipykernel                     5.4.3
60 ipython                       7.20.0
61 ipython-genutils              0.2.0
62 ipywidgets                    6.0.0
63 jedi                          0.18.0
64 Jinja2                        2.11.3
65 joblib                        0.17.0
66 jsonschema                    3.2.0
67 jupyter-client                6.1.11
68 jupyter-console               6.2.0
69 jupyter-core                  4.7.1
70 keras                         3.7.0
71 Keras-Preprocessing           1.1.2
72 kiwisolver                    1.4.7
73 langcodes                     3.5.0
74 language-data                 1.3.0
75 libclang                      18.1.1
76 lockfile                      0.12.2
77 louis                         3.16.0
78 lxml                          4.6.3
```

```
79  mail-parser                  4.1.2
80  Mako                         1.1.3
81  marisa-trie                  1.2.1
82  Markdown                     3.3.4
83  markdown-it-py               3.0.0
84  MarkupSafe                   1.1.1
85  matplotlib                   3.4.3
86  mdurl                        0.1.2
87  mistune                      0.8.4
88  ml-dtypes                    0.4.1
89  monotonic                    1.5
90  more-itertools               4.2.0
91  mpmath                       1.3.0
92  murmurhash                   1.0.11
93  namex                        0.0.8
94  nbconvert                    5.6.1
95  nbformat                     5.1.2
96  networkx                     3.2.1
97  nilsimsa                     0.3.8
98  nltk                         3.5
99  nose                         1.3.7
100 notebook                     6.2.0
101 numpy                        1.19.5
102 nvidia-cublas-cu12           12.4.5.8
103 nvidia-cuda-cupti-cu12       12.4.127
104 nvidia-cuda-nvrtc-cu12       12.4.127
105 nvidia-cuda-runtime-cu12     12.4.127
106 nvidia-cudnn-cu12            9.1.0.70
107 nvidia-cufft-cu12            11.2.1.3
108 nvidia-curand-cu12           10.3.5.147
109 nvidia-cusolver-cu12         11.6.1.9
110 nvidia-cusparse-cu12         12.3.1.170
111 nvidia-nccl-cu12             2.21.5
112 nvidia-nvjitlink-cu12        12.4.127
113 nvidia-nvtx-cu12             12.4.127
114 oauthlib                     3.1.0
115 olefile                      0.46
116 opt-einsum                   3.3.0
117 optree                       0.13.1
```

```
118 packaging                    20.9
119 pandas                       1.3.3
120 pandocfilters                1.4.3
121 parameterized                0.7.0
122 paramiko                     2.7.2
123 parso                        0.8.1
124 pexpect                      4.8.0
125 pickleshare                  0.7.5
126 pillow                       11.0.0
127 pip                          20.3.4
128 platformdirs                 4.3.6
129 preshed                      3.0.9
130 prometheus-client            0.9.0
131 prompt-toolkit               3.0.14
132 protobuf                     3.20.0
133 psutil                       5.8.0
134 pyasn1                       0.6.1
135 pyasn1-modules               0.4.1
136 pybind11                     2.13.6
137 pycairo                      1.16.2
138 pycountry                    24.6.1
139 pycparser                    2.22
140 pycups                       2.0.1
141 pycurl                       7.43.0.6
142 pydantic                     2.10.3
143 pydantic-core                2.27.1
144 pydot                        3.0.3
145 pydotplus                    2.0.2
146 pygments                     2.18.0
147 PyGObject                    3.38.0
148 pygpu                        0.7.6
149 pyinotify                    0.9.6
150 PyJWT                        1.7.1
151 PyNaCl                       1.4.0
152 pyparsing                    3.2.0
153 pyrsistent                   0.15.5
154 PySimpleSOAP                 1.16.2
155 pysmbc                       1.0.23
156 python-apt                   2.2.1
```

```
157 python-dateutil            2.9.0.post0
158 python-debian              0.1.39
159 python-debianbts           3.1.0
160 python-pam                 1.8.4
161 python-xapp                2.0.2
162 python-xlib                0.29
163 pytz                       2021.1
164 pyxdg                      0.27
165 PyYAML                     5.3.1
166 pyzmq                      20.0.0
167 regex                      2020.11.13
168 requests                   2.25.1
169 requests-oauthlib          2.0.0
170 rich                       13.9.4
171 rsa                        4.9
172 sacremoses                 0.0.53
173 scikit-learn               1.0
174 scipy                      1.6.0
175 scour                      0.38.2
176 seaborn                    0.13.2
177 Send2Trash                 1.6.0b1
178 sentencepiece              0.2.0
179 setproctitle               1.2.1
180 setuptools                 52.0.0
181 shellingham                1.5.4
182 simplejson                 3.19.3
183 six                        1.15.0
184 smart-open                 7.0.5
185 soupsieve                  2.2.1
186 spacy                      3.8.2
187 spacy-legacy               3.0.12
188 spacy-loggers              1.0.5
189 srsly                      2.4.8
190 stanza                     1.1.1
191 sympy                      1.13.1
192 tensorboard                2.18.0
193 tensorboard-data-server    0.7.2
194 tensorboard-plugin-wit     1.8.0
195 tensorflow                 2.18.0
```

```
196 tensorflow-estimator           2.6.0
197 tensorflow-hub                 0.16.1
198 tensorflow-io-gcs-filesystem   0.37.1
199 tensorflow-text                2.18.0
200 termcolor                      1.1.0
201 terminado                      0.9.2
202 testpath                       0.4.4
203 tf-keras                       2.18.0
204 Theano                         1.0.5
205 thinc                          8.3.2
206 threadpoolctl                  3.5.0
207 tinycss                        0.4
208 tinycss2                       1.0.2
209 torch                          2.5.1
210 tornado                        6.1
211 tqdm                           4.57.0
212 traitlets                      5.0.5
213 triton                         3.1.0
214 typer                          0.15.1
215 typing-extensions              4.12.2
216 tzdata                         2024.2
217 urllib3                        1.26.5
218 virtualenv                     20.28.0
219 visidata                       3.1.1
220 wasabi                         1.1.3
221 wcwidth                        0.1.9
222 weasel                         0.4.1
223 webencodings                   0.5.1
224 werkzeug                       3.1.3
225 wheel                          0.45.1
226 widgetsnbextension             2.0.0
227 wordcloud                      1.9.4
228 wrapt                          1.12.1
229 xdg                            5
230 zipp                           3.21.0
```

Listing 21: notebook.pip

B.3. desktop.pip

```
annotated-types          0.7.0
argon2-cffi              18.3.0
argostranslate           1.9.6
attrs                    20.3.0
backcall                 0.2.0
beautifulsoup4           4.12.3
bleach                   3.2.1
blis                     1.0.1
catalogue                2.0.10
certifi                  2024.8.30
charset-normalizer       3.4.0
click                    8.1.7
cloudpathlib             0.20.0
confection               0.1.5
ctranslate2              4.5.0
cupy-cuda12x             13.3.0
curated-tokenizers       0.0.9
curated-transformers     0.1.1
cycler                   0.12.1
cymem                    2.0.8
dbus-python              1.2.16
de_core_news_md          3.8.0
de_dep_news_trf          3.8.0
decorator                4.4.2
defusedxml               0.6.0
distlib                  0.3.9
en_core_web_md           3.8.0
en_core_web_trf          3.8.0
entrypoints              0.3
fastrlock                0.8.2
fasttext                 0.9.3
filelock                 3.16.1
fsspec                   2024.10.0
```

```
34 html5lib                    1.1
35 idna                        3.10
36 importlib_metadata          8.5.0
37 ipykernel                   5.4.3
38 ipython                     7.20.0
39 ipython_genutils            0.2.0
40 jedi                        0.18.0
41 Jinja2                      2.11.3
42 joblib                      1.4.2
43 jsonschema                  3.2.0
44 jupyter-client              6.1.11
45 jupyter-core                4.7.1
46 kiwisolver                  1.4.7
47 langcodes                   3.4.1
48 language_data               1.2.0
49 lxml                        4.6.3
50 marisa-trie                 1.2.1
51 Markdown                    3.3.4
52 markdown-it-py              3.0.0
53 MarkupSafe                  1.1.1
54 matplotlib                  3.4.3
55 mdurl                       0.1.2
56 mistune                     0.8.4
57 more-itertools              4.2.0
58 mpmath                      1.3.0
59 murmurhash                  1.0.10
60 nbconvert                   5.6.1
61 nbformat                    5.1.2
62 networkx                    3.2.1
63 notebook                    6.2.0
64 numpy                       1.22.4
65 nvidia-cublas-cu12          12.4.5.8
66 nvidia-cuda-cupti-cu12      12.4.127
67 nvidia-cuda-nvrtc-cu12      12.4.127
68 nvidia-cuda-runtime-cu12    12.4.127
69 nvidia-cudnn-cu12           9.1.0.70
70 nvidia-cufft-cu12           11.2.1.3
71 nvidia-curand-cu12          10.3.5.147
72 nvidia-cusolver-cu12        11.6.1.9
```

```
73  nvidia-cusparse-cu12        12.3.1.170
74  nvidia-nccl-cu12            2.21.5
75  nvidia-nvjitlink-cu12       12.4.127
76  nvidia-nvtx-cu12            12.4.127
77  packaging                   20.9
78  pandas                      1.3.3
79  pandocfilters               1.4.3
80  parso                       0.8.1
81  pexpect                     4.8.0
82  pickleshare                 0.7.5
83  pillow                      11.0.0
84  pip                         24.3.1
85  platformdirs                4.3.6
86  preshed                     3.0.9
87  prometheus-client           0.9.0
88  prompt-toolkit              3.0.14
89  protobuf                    3.20.0
90  pybind11                    2.13.6
91  pycurl                      7.43.0.6
92  pydantic                    2.9.2
93  pydantic_core               2.23.4
94  Pygments                    2.18.0
95  PyGObject                   3.38.0
96  pyparsing                   2.4.7
97  pyrsistent                  0.15.5
98  python-apt                  2.2.1
99  python-dateutil             2.9.0.post0
100 pytz                        2024.2
101 PyYAML                      6.0.2
102 pyzmq                       20.0.0
103 regex                       2024.9.11
104 requests                    2.32.3
105 rich                        13.9.3
106 sacremoses                  0.0.53
107 Send2Trash                  1.6.0b1
108 sentencepiece               0.2.0
109 setuptools                  75.2.0
110 shellingham                 1.5.4
111 six                         1.16.0
```

```
112 smart-open                   7.0.5
113 soupsieve                    2.6
114 spacy                        3.8.2
115 spacy-curated-transformers 0.3.0
116 spacy-legacy                 3.0.12
117 spacy-loggers                1.0.5
118 srsly                        2.4.8
119 stanza                       1.1.1
120 sympy                        1.13.1
121 terminado                    0.9.2
122 testpath                     0.4.4
123 thinc                        8.3.2
124 torch                        2.5.0
125 tornado                      6.1
126 tqdm                         4.66.5
127 traitlets                    5.0.5
128 triton                       3.1.0
129 typer                        0.12.5
130 typing_extensions            4.12.2
131 tzdata                       2024.2
132 urllib3                      2.2.3
133 virtualenv                   20.27.0
134 visidata                     3.1.1
135 wasabi                       1.1.3
136 wcwidth                      0.1.9
137 weasel                       0.4.1
138 webencodings                 0.5.1
139 wheel                        0.34.2
140 wrapt                        1.16.0
141 xx_ent_wiki_sm               3.8.0
142 xx_sent_ud_sm                3.8.0
143 zipp                         3.20.2
```

Listing 22: desktop.pip

B.4. translate.pip

```
annotated-types          0.7.0
argon2-cffi              18.3.0
argostranslate           1.9.6
attrs                    20.3.0
backcall                 0.2.0
beautifulsoup4           4.12.3
bleach                   3.2.1
blis                     0.7.11
catalogue                2.0.10
certifi                  2024.8.30
charset-normalizer       3.4.0
click                    8.1.7
cloudpathlib             0.20.0
confection               0.1.5
ctranslate2              4.5.0
cupy-cuda12x             13.3.0
cycler                   0.12.1
cymem                    2.0.10
dbus-python              1.2.16
decorator                4.4.2
defusedxml               0.6.0
distlib                  0.3.9
entrypoints              0.3
fastrlock                0.8.2
fasttext                 0.9.3
filelock                 3.16.1
fsspec                   2024.10.0
html5lib                 1.1
idna                     3.10
importlib_metadata       8.5.0
ipykernel                5.4.3
ipython                  7.20.0
ipython_genutils         0.2.0
jedi                     0.18.0
Jinja2                   2.11.3
joblib                   1.4.2
jsonschema               3.2.0
```

```
38  jupyter-client           6.1.11
39  jupyter-core             4.7.1
40  kiwisolver               1.4.7
41  langcodes                3.5.0
42  language_data            1.3.0
43  lxml                     4.6.3
44  marisa-trie              1.2.1
45  Markdown                 3.3.4
46  markdown-it-py           3.0.0
47  MarkupSafe               1.1.1
48  matplotlib               3.4.3
49  mdurl                    0.1.2
50  mistune                  0.8.4
51  more-itertools           4.2.0
52  mpmath                   1.3.0
53  murmurhash               1.0.11
54  nbconvert                5.6.1
55  nbformat                 5.1.2
56  networkx                 3.2.1
57  notebook                 6.2.0
58  numpy                    1.22.4
59  nvidia-cublas-cu12       12.4.5.8
60  nvidia-cuda-cupti-cu12   12.4.127
61  nvidia-cuda-nvrtc-cu12   12.4.127
62  nvidia-cuda-runtime-cu12 12.4.127
63  nvidia-cudnn-cu12        9.1.0.70
64  nvidia-cufft-cu12        11.2.1.3
65  nvidia-curand-cu12       10.3.5.147
66  nvidia-cusolver-cu12     11.6.1.9
67  nvidia-cusparse-cu12     12.3.1.170
68  nvidia-nccl-cu12         2.21.5
69  nvidia-nvjitlink-cu12    12.4.127
70  nvidia-nvtx-cu12         12.4.127
71  packaging                20.9
72  pandas                   1.3.3
73  pandocfilters            1.4.3
74  parso                    0.8.1
75  pexpect                  4.8.0
76  pickleshare              0.7.5
```

```
77  pillow                     11.0.0
78  pip                        24.3.1
79  platformdirs               4.3.6
80  preshed                    3.0.9
81  prometheus-client          0.9.0
82  prompt-toolkit             3.0.14
83  protobuf                   3.20.0
84  pybind11                   2.13.6
85  pycurl                     7.43.0.6
86  pydantic                   2.10.2
87  pydantic_core              2.27.1
88  Pygments                   2.18.0
89  PyGObject                  3.38.0
90  pyparsing                  2.4.7
91  pyrsistent                 0.15.5
92  python-apt                 2.2.1
93  python-dateutil            2.9.0.post0
94  pytz                       2024.2
95  PyYAML                     6.0.2
96  pyzmq                      20.0.0
97  regex                      2024.11.6
98  requests                   2.32.3
99  rich                       13.9.4
100 sacremoses                 0.0.53
101 Send2Trash                 1.6.0b1
102 sentencepiece              0.2.0
103 setuptools                 75.6.0
104 shellingham                1.5.4
105 six                        1.16.0
106 smart-open                 7.0.5
107 soupsieve                  2.6
108 spacy                      3.8.0
109 spacy-legacy               3.0.12
110 spacy-loggers              1.0.5
111 srsly                      2.4.8
112 stanza                     1.1.1
113 sympy                      1.13.1
114 terminado                  0.9.2
115 testpath                   0.4.4
```

```
116 thinc                    8.2.5
117 torch                    2.5.1
118 tornado                  6.1
119 tqdm                     4.67.1
120 traitlets                5.0.5
121 triton                   3.1.0
122 typer                    0.13.1
123 typing_extensions        4.12.2
124 tzdata                   2024.2
125 urllib3                  2.2.3
126 virtualenv               20.28.0
127 visidata                 3.1.1
128 wasabi                   1.1.3
129 wcwidth                  0.1.9
130 weasel                   0.4.1
131 webencodings             0.5.1
132 wheel                    0.34.2
133 wrapt                    1.17.0
134 xx_ent_wiki_sm           3.8.0
135 xx_sent_ud_sm            3.8.0
136 zipp                     3.21.0
```

Listing 23: translate.pip

B.5. rtx2070.pip

```
1 absl-py                  0.14.1
2 appdirs                  1.4.4
3 argon2-cffi              18.3.0
4 astunparse               1.6.3
5 attrs                    20.3.0
6 backcall                 0.2.0
7 bcrypt                   3.1.7
8 beautifulsoup4           4.9.3
```

```
 9  bleach                       3.2.1
10  blinker                      1.4
11  Brlapi                       0.8.2
12  cachetools                   4.2.4
13  certifi                      2020.6.20
14  cffi                         1.15.1
15  chardet                      4.0.0
16  clang                        5.0
17  cryptography                 3.3.2
18  cupshelpers                  1.0
19  cycler                       0.10.0
20  Cython                       0.29.21
21  dbus-python                  1.2.16
22  decorator                    4.4.2
23  defusedxml                   0.6.0
24  distlib                      0.3.1
25  distro                       1.5.0
26  distro-info                  1.0
27  duplicity                    0.8.17
28  entrypoints                  0.3
29  fail2ban                     0.11.2
30  fasteners                    0.14.1
31  fasttext                     0.9.2
32  filelock                     3.0.12
33  flatbuffers                  1.12
34  future                       0.18.2
35  gast                         0.4.0
36  google-auth                  1.35.0
37  google-auth-oauthlib         0.4.6
38  google-pasta                 0.2.0
39  grpcio                       1.41.0
40  h5py                         3.1.0
41  h5py.-debian-h5py-serial     2.10.0
42  html5lib                     1.1
43  httplib2                     0.18.1
44  idna                         2.10
45  importlib-metadata           1.6.0
46  invoke                       1.4.1
47  ipykernel                    5.4.3
```

```
48 ipython                  7.20.0
49 ipython-genutils         0.2.0
50 ipywidgets               6.0.0
51 jedi                     0.18.0
52 Jinja2                   2.11.3
53 joblib                   1.1.0
54 jsonschema               3.2.0
55 jupyter-client           6.1.11
56 jupyter-console          6.2.0
57 jupyter-core             4.7.1
58 keras                    2.6.0
59 Keras-Applications       1.0.8
60 Keras-Preprocessing      1.1.2
61 kiwisolver               1.3.2
62 lockfile                 0.12.2
63 louis                    3.16.0
64 lxml                     4.6.3
65 mail-parser              3.15.0
66 Mako                     1.1.3
67 Markdown                 3.3.4
68 MarkupSafe               1.1.1
69 matplotlib               3.4.3
70 mistune                  0.8.4
71 monotonic                1.5
72 more-itertools           4.2.0
73 nbconvert                5.6.1
74 nbformat                 5.1.2
75 nose                     1.3.7
76 notebook                 6.2.0
77 numpy                    1.19.5
78 oauthlib                 3.1.0
79 olefile                  0.46
80 opt-einsum               3.3.0
81 packaging                20.9
82 pandas                   1.3.3
83 pandocfilters            1.4.3
84 parameterized            0.7.0
85 paramiko                 2.7.2
86 parso                    0.8.1
```

```
87  pexpect                  4.8.0
88  pickleshare              0.7.5
89  Pillow                   8.1.2
90  pip                      20.3.4
91  prometheus-client        0.9.0
92  prompt-toolkit           3.0.14
93  protobuf                 3.18.1
94  psutil                   5.8.0
95  ptyprocess               0.7.0
96  pyasn1                   0.4.8
97  pyasn1-modules           0.2.8
98  pybind11                 2.8.1
99  pycairo                  1.16.2
100 pycountry                20.7.3
101 pycparser                2.21
102 pycups                   2.0.1
103 pycurl                   7.43.0.6
104 pydot                    1.4.2
105 pydotplus                2.0.2
106 Pygments                 2.16.1
107 PyGObject                3.38.0
108 pyinotify                0.9.6
109 PyJWT                    1.7.1
110 PyNaCl                   1.4.0
111 pyparsing                2.4.7
112 pyrsistent               0.15.5
113 PySimpleSOAP             1.16.2
114 pysmbc                   1.0.23
115 python-apt               2.2.1
116 python-dateutil          2.8.1
117 python-debian            0.1.39
118 python-debianbts         3.1.0
119 python-pam               1.8.4
120 python-xapp              2.0.2
121 python-xlib              0.29
122 pytz                     2021.1
123 pyxdg                    0.27
124 PyYAML                   5.3.1
125 pyzmq                    20.0.0
```

```
126 requests                  2.25.1
127 requests-oauthlib         1.3.0
128 rsa                       4.7.2
129 scikit-learn              1.0
130 scipy                     1.6.0
131 scour                     0.38.2
132 seaborn                   0.11.2
133 Send2Trash                1.6.0b1
134 setproctitle              1.2.1
135 setuptools                68.1.2
136 simplejson                3.17.5
137 six                       1.15.0
138 sklearn                   0.0
139 soupsieve                 2.2.1
140 systemd-python            234
141 tensorboard               2.6.0
142 tensorboard-data-server   0.6.1
143 tensorboard-plugin-wit    1.8.0
144 tensorflow                2.6.0
145 tensorflow-estimator      2.6.0
146 tensorflow-gpu            2.6.0
147 termcolor                 1.1.0
148 terminado                 0.9.2
149 testpath                  0.4.4
150 Theano                    1.0.5
151 threadpoolctl             3.0.0
152 tinycss                   0.4
153 tinycss2                  1.0.2
154 tornado                   6.1
155 traitlets                 5.0.5
156 typing-extensions         3.7.4.3
157 unattended-upgrades       0.1
158 urllib3                   1.26.5
159 virtualenv                20.4.0+ds
160 visidata                  2.2.1
161 wcwidth                   0.1.9
162 webencodings              0.5.1
163 Werkzeug                  2.0.2
164 wheel                     0.41.2
```

```
165 widgetsnbextension        2.0.0
166 wordcloud                 1.8.1
167 wrapt                     1.12.1
168 xdg                       5.0.0
169 zipp                      1.0.0
```

Listing 24: x2070.pip

B.6. llm.pip

```
1  accelerate                  1.1.1
2  aiohappyeyeballs            2.4.4
3  aiohttp                     3.11.9
4  aiosignal                   1.3.1
5  anyio                       4.6.2.post1
6  argon2-cffi                 23.1.0
7  argon2-cffi-bindings        21.2.0
8  arrow                       1.3.0
9  asttokens                   3.0.0
10 async-lru                   2.0.4
11 async-timeout               5.0.1
12 attrs                       24.2.0
13 babel                       2.16.0
14 beautifulsoup4              4.12.3
15 bitsandbytes                0.44.1
16 bleach                      6.2.0
17 certifi                     2024.8.30
18 cffi                        1.17.1
19 charset-normalizer          3.4.0
20 comm                        0.2.2
21 debugpy                     1.8.9
22 decorator                   5.1.1
23 defusedxml                  0.7.1
24 einops                      0.8.0
```

```
exceptiongroup              1.2.2
executing                   2.1.0
fastjsonschema              2.21.1
filelock                    3.16.1
flash-attn                  2.7.0.post2
fqdn                        1.5.1
frozenlist                  1.5.0
fsspec                      2024.10.0
h11                         0.14.0
httpcore                    1.0.7
httpx                       0.28.0
huggingface-hub             0.26.3
idna                        3.10
ipykernel                   6.29.5
ipython                     8.30.0
ipywidgets                  8.1.5
isoduration                 20.11.0
jedi                        0.19.2
Jinja2                      3.1.4
joblib                      1.4.2
json5                       0.10.0
jsonpointer                 3.0.0
jsonschema                  4.23.0
jsonschema-specifications   2024.10.1
jupyter                     1.1.1
jupyter_client              8.6.3
jupyter-console             6.6.3
jupyter_core                5.7.2
jupyter-events              0.10.0
jupyter-lsp                 2.2.5
jupyter_server              2.14.2
jupyter_server_terminals    0.5.3
jupyterlab                  4.2.6
jupyterlab_pygments         0.3.0
jupyterlab_server           2.27.3
jupyterlab_widgets          3.0.13
lightning                   2.4.0
lightning-utilities         0.11.9
MarkupSafe                  3.0.2
```

```
64  matplotlib-inline         0.1.7
65  mistune                   3.0.2
66  mpmath                    1.3.0
67  multidict                 6.1.0
68  nbclient                  0.10.1
69  nbconvert                 7.16.4
70  nbformat                  5.10.4
71  nest-asyncio              1.6.0
72  networkx                  3.4.2
73  notebook                  7.2.2
74  notebook_shim             0.2.4
75  numpy                     2.1.3
76  nvidia-cublas-cu12        12.1.3.1
77  nvidia-cuda-cupti-cu12    12.1.105
78  nvidia-cuda-nvrtc-cu12    12.1.105
79  nvidia-cuda-runtime-cu12  12.1.105
80  nvidia-cudnn-cu12         8.9.2.26
81  nvidia-cufft-cu12         11.0.2.54
82  nvidia-curand-cu12        10.3.2.106
83  nvidia-cusolver-cu12      11.4.5.107
84  nvidia-cusparse-cu12      12.1.0.106
85  nvidia-nccl-cu12          2.20.5
86  nvidia-nvjitlink-cu12     12.6.85
87  nvidia-nvtx-cu12          12.1.105
88  overrides                 7.7.0
89  packaging                 24.2
90  pandas                    2.2.3
91  pandocfilters             1.5.1
92  parso                     0.8.4
93  pathlib                   1.0.1
94  pexpect                   4.9.0
95  pip                       22.0.2
96  platformdirs              4.3.6
97  prometheus_client         0.21.0
98  prompt_toolkit            3.0.48
99  propcache                 0.2.1
100 psutil                    6.1.0
101 ptyprocess                0.7.0
102 pure_eval                 0.2.3
```

```
103 pycparser                 2.22
104 Pygments                  2.18.0
105 python-dateutil           2.9.0.post0
106 python-json-logger        2.0.7
107 pytorch-lightning         2.4.0
108 pytz                      2024.2
109 PyYAML                    6.0.2
110 pyzmq                     26.2.0
111 referencing               0.35.1
112 regex                     2024.11.6
113 requests                  2.32.3
114 rfc3339-validator         0.1.4
115 rfc3986-validator         0.1.1
116 rpds-py                   0.21.0
117 safetensors               0.4.5
118 scikit-learn              1.5.2
119 scipy                     1.14.1
120 Send2Trash                1.8.3
121 sentencepiece             0.2.0
122 setuptools                59.6.0
123 six                       1.16.0
124 sniffio                   1.3.1
125 soupsieve                 2.6
126 stack-data                0.6.3
127 sympy                     1.13.3
128 terminado                 0.18.1
129 threadpoolctl             3.5.0
130 tinycss2                  1.4.0
131 tokenizers                0.20.3
132 tomli                     2.2.1
133 torch                     2.3.0
134 torchmetrics              1.6.0
135 tornado                   6.4.2
136 tqdm                      4.67.1
137 traitlets                 5.14.3
138 transformers              4.46.3
139 triton                    2.3.0
140 types-python-dateutil     2.9.0.20241003
141 typing_extensions         4.12.2
```

```
142  tzdata                    2024.2
143  uri-template              1.3.0
144  urllib3                   2.2.3
145  wcwidth                   0.2.13
146  webcolors                 24.11.1
147  webencodings              0.5.1
148  websocket-client          1.8.0
149  wheel                     0.37.1
150  widgetsnbextension        4.0.13
151  yarl                      1.18.3
```

Listing 25: llm.pip

C. Anhang Containerfiles

C.1. Containerfile.jupyter-notebook

```
FROM debian:bullseye-slim

ARG VERSION=0.29

ENV LANG C
ENV DEBIAN_FRONTEND noninteractive

# hosts fuer abcupdate
RUN echo '10.25.175.10 abcupdate abcupdate.abc.at' » /etc/hosts
 ↪ && \
        echo 'deb http://abcupdate.abc.at/debian/ bullseye main
         ↪ contrib non-free' > /etc/apt/sources.list && \
        echo 'deb http://abcupdate.abc.at/debian/
         ↪ bullseye-updates main contrib non-free' »
         ↪ /etc/apt/sources.list && \
        echo 'deb
         ↪ http://abcupdate.abc.at/security/debian-security
         ↪ bullseye-security main contrib' »
         ↪ /etc/apt/sources.list

# qtbase5-dev skipped
RUN apt-get update && \
    apt-get upgrade -y && \
    apt-get dist-upgrade -y && \
    apt-get install -y -no-install-recommends sudo && \
    apt-get install -y -no-install-recommends jupyter-notebook &&
     ↪ \
    apt-get install -y -no-install-recommends libpython3-dev
     ↪ libpython3-stdlib libpython3.9-dev libpython3.9-minimal
     ↪ libpython3.9-stdlib libpython3.9 python-apt-common
     ↪ python-tinycss2-common python3-appdirs python3-apt
     ↪ python3-argon2 python3-attr python3-backcall
     ↪ python3-bcrypt python3-bleach python3-blinker python3-bs4
```

```
21      apt-get install -y -no-install-recommends python3-distlib
         ↪  python3-distro-info python3-distro python3-distutils
         ↪  python3-entrypoints python3-fasteners python3-filelock
         ↪  python3-future python3-gi-cairo python3-gi
         ↪  python3-h5py-serial python3-h5py python3-html5lib
         ↪  python3-httplib2 python3-idna python3-importlib-metadata
         ↪  python3-invoke
22          apt-get install -y -no-install-recommends
             ↪  python3-ipykernel python3-ipython-genutils
             ↪  python3-ipywidgets python3-jedi python3-jinja2
             ↪  python3-jsonschema python3-jupyter-client
             ↪  python3-jupyter-console python3-jwt python3-lib2to3
             ↪  python3-lockfile python3-louis python3-mako
             ↪  python3-minimal python3-mistune python3-monotonic
             ↪  python3-more-itertools python3-nacl python3-nbconvert
             ↪  python3-nbformat python3-nose python3-oauthlib
23          apt-get install -y -no-install-recommends python3-olefile
             ↪  python3-packaging python3-pampy python3-pandocfilters
             ↪  python3-parameterized python3-paramiko python3-parso
             ↪  python3-pexpect python3-pickleshare
             ↪  python3-pkg-resources python3-prometheus-client
             ↪  python3-prompt-toolkit python3-psutil
             ↪  python3-ptyprocess python3-pyatspi python3-pycurl
             ↪  python3-pygments python3-pygpu python3-pyinotify
             ↪  python3-pyparsing python3-pyrsistent
             ↪  python3-pysimplesoap python3-requests python3-scipy
24          apt-get install -y -no-install-recommends python3-scour
             ↪  python3-send2trash python3-setproctitle
             ↪  python3-setuptools python3-six python3-smbc
             ↪  python3-software-properties python3-soupsieve
             ↪  python3-talloc python3-terminado python3-testpath
             ↪  python3-theano python3-tinycss2 python3-tinycss
             ↪  python3-tornado python3-traitlets python3-tz
             ↪  python3-urllib3 python3-wcwidth python3-webencodings
             ↪  python3-wheel python3-widgetsnbextension python3-xapp
             ↪  python3-xdg python3-xlib python3-yaml python3-zipp
             ↪  python3-zmq python3.9-dev python3.9-minimal python3.9
             ↪  python3 && \
25      apt-get install -y -no-install-recommends pandoc python3-nltk
         ↪  && \
```

```
26      apt-get install -y python3-minimal python3-pip wget && \
27      apt-get install -y -no-install-recommends ca-certificates &&
         ↪  \
28      apt-get install -y -no-install-recommends
         ↪  software-properties-common && \
29      apt-get clean && \
30      rm -rf /var/lib/apt/lists/*
31
32  ADD ./certs/msca11.crt
     ↪  /usr/local/share/ca-certificates/msca11.crt
33  RUN chmod 644 /usr/local/share/ca-certificates/msca11.crt &&
     ↪  update-ca-certificates
34
35  RUN python3 -m pip install tensorflow==2.6.0 keras==2.6.0 &&
     ↪  python3 -m pip cache purge
36  RUN python3 -m pip install Theano==1.0.5 tensorboard==2.6.0
     ↪  tensorboard-data-server==0.6.1 tensorboard-plugin-wit==1.8.0
     ↪  tensorflow-estimator==2.6.0 && python3 -m pip cache purge
37  RUN python3 -m pip install scikit-learn==1.0 notebook==6.2.0
     ↪  jupyter-client==6.1.11 jupyter-console==6.2.0
     ↪  jupyter-core==4.7.1 && python3 -m pip cache purge
38  RUN python3 -m pip install absl-py argon2-cffi astunparse attrs
     ↪  beautifulsoup4 cachetools cffi clang cycler Cython fasteners
     ↪  fasttext flatbuffers gast google-auth google-auth-oauthlib
     ↪  google-pasta grpcio httplib2 ipython ipython-genutils Jinja2
     ↪  joblib kiwisolver lxml mail-parser Mako opt-einsum && python3
     ↪  -m pip cache purge
39
40  RUN python3 -m pip install Pillow pyasn1 pyasn1-modules pybind11
     ↪  pycountry pycparser pycups pydot && python3 -m pip cache
     ↪  purge
41  RUN python3 -m pip install pydotplus Pygments PyGObject PyJWT
     ↪  PyNaCl PySimpleSOAP pysmbc python-pam pytz pyxdg PyYAML &&
     ↪  python3 -m pip cache purge
42  RUN python3 -m pip install pyzmq rsa seaborn Send2Trash
     ↪  simplejson termcolor threadpoolctl typing-extensions
     ↪  virtualenv wordcloud wrapt && python3 -m pip cache purge
43  RUN python3 -m pip install argostranslate spacy torch && python3
     ↪  -m pip cache purge
```

```
44 RUN python3 -m pip install tensorflow_hub tensorflow_text
   ↪ nilsimsa
45 RUN python3 -m pip install visidata Werkzeug
46 # alles letztes dann hoffentlich OK
47 RUN python3 -m pip install Markdown==3.3.4 MarkupSafe==1.1.1
   ↪ protobuf==3.20 numpy>=1.20 pandas==1.3.3 Matplotlib==3.4.3 &&
   ↪ python3 -m pip cache purge
48
49 #Install CUDA Treiber
50 RUN wget
   ↪ https://developer.download.nvidia.com/compute/cuda/repos/
   ↪ debian11/x86_64/cuda-keyring_1.1-1_all.deb && \
51         dpkg -i cuda-keyring_1.1-1_all.deb && \
52         add-apt-repository contrib && \
53         apt-get update && \
54         apt-get -y install cuda-toolkit-12-6 && \
55         apt-get -y install libcudnn8 && \
56         apt-get clean && \
57     rm -rf /var/lib/apt/lists/*
58
59 # Create jyupter user and tini fuer jupyter
60 RUN useradd -ms /bin/bash jupyter
61 RUN passwd -d jupyter
62 RUN printf 'jupyter ALL=(ALL) ALL\n' | tee -a /etc/sudoers
63
64 USER jupyter
65 WORKDIR "/home/jupyter"
66
67 CMD ["/bin/bash"]
68 CMD ["jupyter", "notebook", "-port=8889", "-no-browser",
   ↪ "-ip=0.0.0.0", "-NotebookApp.token=''"]
69
70 LABEL \
71     org.opencontainers.image.vendor="ABC" \
72     org.opencontainers.image.title="Jupyter Notebook" \
73     org.opencontainers.image.description="Jupyter Notebook with
       ↪ GPU support based on Debian Classic" \
74     org.opencontainers.image.version="${VERSION}" \
75     org.opencontainers.image.url="https://jupyter.org/" \
```

```
    org.opencontainers.image.source="https://git02.abc.at/containers/podman-build/"
```

Listing 26: containerfile.jupyter-notebookInput

C.2. Containerfile.translate

```
FROM debian:bullseye-slim

ARG VERSION=0.39

ENV LANG C
ENV DEBIAN_FRONTEND noninteractive
ENV ARGOS_DEVICE_TYPE=cuda

# hosts fuer abcupdate
RUN echo '10.25.175.10 abcupdate abcupdate.abc.at' » /etc/hosts && \
        echo 'deb http://abcupdate.abc.at/debian/ bullseye main contrib non-free' > /etc/apt/sources.list && \
        echo 'deb http://abcupdate.abc.at/debian/ bullseye-updates main contrib non-free' » /etc/apt/sources.list && \
        echo 'deb http://abcupdate.abc.at/security/debian-security bullseye-security main contrib' » /etc/apt/sources.list

# qtbase5-dev skipped
RUN apt-get update && \
    apt-get upgrade -y && \
    apt-get dist-upgrade -y && \
    apt-get install -y -no-install-recommends sudo wget software-properties-common && \
```

```
20      apt-get install -y -no-install-recommends jupyter-notebook &&
         ↪  \
21          apt-get install -y python3-minimal python3-pip wget && \
22          apt-get install -y -no-install-recommends ca-certificates
             ↪  && \
23          apt-get install -y -no-install-recommends
             ↪  software-properties-common && \
24      apt-get clean && \
25      rm -rf /var/lib/apt/lists/*
26
27  ADD ./certs/msca11.crt
     ↪  /usr/local/share/ca-certificates/msca11.crt
28  RUN chmod 644 /usr/local/share/ca-certificates/msca11.crt &&
     ↪  update-ca-certificates
29
30
31  RUN python3 -m pip install virtualenv beautifulsoup4 fasttext
     ↪  ipython pandas visidata
32  RUN python3 -m pip install argostranslate
33  RUN python3 -m pip install -U setuptools pip # muss for cupy
     ↪  passieren sonst compiliert das nie
34  RUN python3 -m pip install cupy-cuda12x # numpy is included in
     ↪  einer zu neuenn Version! fuer meine for schleife hab dort
     ↪  1.19.5
35  RUN python3 -m pip install spacy==3.8.0
36  RUN python3 -m pip install torch
37  RUN python3 -m pip install Markdown==3.3.4 MarkupSafe==1.1.1
     ↪  protobuf==3.20 numpy==1.22.4 pandas==1.3.3 Matplotlib==3.4.3
     ↪  # numpy so wie ich numpy verwende
38  RUN python3 -m pip install protobuf==3.20 # noetig fuer spacy
     ↪  Downloads
39
40  # Modell fuer Spacy
41  #RUN spacy download "en_core_web_md"
42  #RUN spacy download "de_core_news_md"
43  #RUN spacy download "xx_ent_wiki_sm"
44  RUN python3 -m pip install
     ↪  https://github.com/explosion/spacy-models/releases/download/
     ↪  xx_ent_wiki_sm-3.8.0/xx_ent_wiki_sm-3.8.0-py3-none-any.whl
```

```
45  #RUN spacy download "en_core_web_trf"
46  #RUN spacy download "de_dep_news_trf"
47  #RUN spacy download "xx_sent_ud_sm"
48  RUN python3 -m pip install
     ↪  https://github.com/explosion/spacy-models/releases/download/
     ↪  xx_sent_ud_sm-3.8.0/xx_sent_ud_sm-3.8.0-py3-none-any.whl
49
50  #Install CUDA Treiber
51  RUN wget
     ↪  https://developer.download.nvidia.com/compute/cuda/repos/
     ↪  debian11/x86_64/cuda-keyring_1.1-1_all.deb && \
52          dpkg -i cuda-keyring_1.1-1_all.deb && \
53          add-apt-repository contrib && \
54          apt-get update && \
55          apt-get -y install cuda-toolkit-12-6 && \
56          apt-get -y install libcudnn8 && \
57          apt-get clean && \
58      rm -rf /var/lib/apt/lists/*
59
60
61  # Create jyupter user and tini fuer jupyter
62  RUN useradd -ms /bin/bash jupyter
63  RUN passwd -d jupyter
64  RUN printf 'jupyter ALL=(ALL) ALL\n' | tee -a /etc/sudoers
65
66  USER jupyter
67  WORKDIR "/home/jupyter"
68
69  CMD ["/bin/bash"]
70  CMD ["jupyter", "notebook", "-port=8888", "-no-browser",
     ↪  "-ip=0.0.0.0", "-NotebookApp.token=''"]
71
72  LABEL \
73      org.opencontainers.image.vendor="ABC" \
74      org.opencontainers.image.title="Jupyter Notebook" \
75      org.opencontainers.image.description="Jupyter Notebook with
          ↪  GPU support based on Debian for Tranalation with Spacy" \
76      org.opencontainers.image.version="${VERSION}" \
77      org.opencontainers.image.url="https://jupyter.org/" \
```

```
78      org.opencontainers.image.source="https://git02.abc.at/
        ↪  containers/podman-build/"
```

Listing 27: containerfile.translate

C.3. Containerfile.llm

```
1   #  podman build -t ubuntu22cuda . -squash
2   FROM docker.io/nvidia/cuda:12.6.2-devel-ubuntu22.04 as base
3
4   ARG VERSION=0.6
5
6   ENV NV_CUDNN_VERSION 9.5.0.50-1
7   ENV NV_CUDNN_PACKAGE_NAME libcudnn9-cuda-12
8   ENV NV_CUDNN_PACKAGE libcudnn9-cuda-12=${NV_CUDNN_VERSION}
9   ENV NV_CUDNN_PACKAGE_DEV
    ↪  libcudnn9-dev-cuda-12=${NV_CUDNN_VERSION}
10
11
12  LABEL maintainer "NVIDIA CORPORATION <cudatools@nvidia.com>"
13  LABEL com.nvidia.cudnn.version="${NV_CUDNN_VERSION}"
14
15  RUN apt-get update && apt-get install -y -no-install-recommends \
16      ${NV_CUDNN_PACKAGE} \
17      ${NV_CUDNN_PACKAGE_DEV} \
18      && apt-mark hold ${NV_CUDNN_PACKAGE_NAME} \
19      && apt install mc python3-pip -y \
20      && apt install sudo findutils iproute2 net-tools procps w3m
        ↪  vim wget -y \
21      && apt install git -y \
22      && rm -rf /var/lib/apt/lists/*
23
24  ADD ./certs/msca11.crt
    ↪  /usr/local/share/ca-certificates/msca11.crt
```

```
RUN chmod 644 /usr/local/share/ca-certificates/msca11.crt && update-ca-certificates

RUN python3 -m pip install torch==2.3 jupyter
RUN python3 -m pip install numpy pandas transformers pathlib tqdm scikit-learn #sklearn
RUN python3 -m pip install sentencepiece accelerate bitsandbytes lightning
RUN python3 -m pip install flash-attn -no-build-isolation

# Create jyupter user and tini fuer jupyter
RUN useradd -ms /bin/bash jupyter
RUN passwd -d jupyter
RUN printf 'jupyter ALL=(ALL) ALL\n' | tee -a /etc/sudoers

USER jupyter
WORKDIR "/home/jupyter"

CMD ["/bin/bash"]
CMD ["jupyter", "notebook", "-port=8888", "-no-browser", "-ip=0.0.0.0", "-NotebookApp.token=''"]

LABEL \

    org.opencontainers.image.vendor="ABC" \
    org.opencontainers.image.title="Jupyter Notebook LLM" \
    org.opencontainers.image.description="Jupyter Notebook with GPU support based on Ubuntu for Traning LLMs" \
    org.opencontainers.image.version="${VERSION}" \
    org.opencontainers.image.url="https://jupyter.org/" \
    org.opencontainers.image.source="https://git02.abc.at/containers/podman-build/"
```

Listing 28: Containerfile.llm

C.4. Dockerfile.LLaMA-Factory

```
# Default use the NVIDIA official image with PyTorch 2.3.0
# https://docs.nvidia.com/deeplearning/frameworks/pytorch-
↪ release-notes/index.html
ARG BASE_IMAGE=nvcr.io/nvidia/pytorch:24.02-py3
FROM ${BASE_IMAGE}

# Define environments
ENV MAX_JOBS=4
ENV FLASH_ATTENTION_FORCE_BUILD=TRUE
ENV VLLM_WORKER_MULTIPROC_METHOD=spawn

# Define installation arguments
ARG INSTALL_BNB=false
ARG INSTALL_VLLM=false
ARG INSTALL_DEEPSPEED=false
ARG INSTALL_FLASHATTN=false
ARG INSTALL_LIGER_KERNEL=false
ARG INSTALL_HQQ=false
ARG INSTALL_EETQ=false
ARG PIP_INDEX=https://pypi.org/simple

# Set the working directory
WORKDIR /app

# Install the requirements
COPY requirements.txt /app
RUN pip config set global.index-url "$PIP_INDEX" && \
    pip config set global.extra-index-url "$PIP_INDEX" && \
    python -m pip install -upgrade pip && \
    python -m pip install -r requirements.txt

# Copy the rest of the application into the image
COPY . /app

# Install the LLaMA Factory
RUN EXTRA_PACKAGES="metrics"; \
```

```
    if [ "$INSTALL_BNB" == "true" ]; then \
        EXTRA_PACKAGES="${EXTRA_PACKAGES},bitsandbytes"; \
    fi; \
    if [ "$INSTALL_VLLM" == "true" ]; then \
        EXTRA_PACKAGES="${EXTRA_PACKAGES},vllm"; \
    fi; \
    if [ "$INSTALL_DEEPSPEED" == "true" ]; then \
        EXTRA_PACKAGES="${EXTRA_PACKAGES},deepspeed"; \
    fi; \
    if [ "$INSTALL_LIGER_KERNEL" == "true" ]; then \
        EXTRA_PACKAGES="${EXTRA_PACKAGES},liger-kernel"; \
    fi; \
    if [ "$INSTALL_HQQ" == "true" ]; then \
        EXTRA_PACKAGES="${EXTRA_PACKAGES},hqq"; \
    fi; \
    if [ "$INSTALL_EETQ" == "true" ]; then \
        EXTRA_PACKAGES="${EXTRA_PACKAGES},eetq"; \
    fi; \
    pip install -e ".[$EXTRA_PACKAGES]"

# Rebuild flash attention
RUN pip uninstall -y transformer-engine flash-attn && \
    if [ "$INSTALL_FLASHATTN" == "true" ]; then \
        pip uninstall -y ninja && pip install ninja && \
        pip install -no-cache-dir flash-attn -no-build-isolation; \
    fi

# Set up volumes
VOLUME [ "/root/.cache/huggingface", "/root/.cache/modelscope", "/app/data", "/app/output" ]

# Expose port 7860 for the LLaMA Board
ENV GRADIO_SERVER_PORT 7860
EXPOSE 7860

# Expose port 8000 for the API service
ENV API_PORT 8000
```

```
72 EXPOSE 8000
```

Listing 29: Dockerfile.LLaMA-Factory.txt