
# Entwicklung einer Syntaxkorrektur im sprachlichen Kontext der Firma HORSCH GmbH

Bachelorarbeit im Fach Medieninformatik am
Institut für Information und Medien, Sprache und Kultur (I:IMSK)

Vorgelegt von: Christoph Meyer
Adresse: Haingrün 24, 95615 Marktredwitz
E-Mail (Universität): christoph.meyer@stud.uni-regensburg.de
E-Mail (privat): christoph.meyer13@hotmail.de
Matrikelnummer: 2036571
Erstgutachter: Prof. Dr. Christian Wolff
Zweitgutachter: Prof. Dr. Nils Henze
Betreuer: Prof. Dr. Christian Wolff
Laufendes Semester: SS2021
Abgegeben am: X

# Inhalt

**1 Einleitung ..... 10**

**2 Related Work ..... 12**

2.1 Neural Machine Translation ..... 12

2.2 Sequence Tagging ..... 17

2.3 Grammatical Error Correction ..... 22

**3 Datenerhebung ..... 25**

3.1 Herkunft der Daten ..... 25

3.2 Extraktion mittels Webcrawler ..... 25

3.2.1 Python-Code ..... 26

3.3 Vorverarbeitung ..... 27

3.3.1 Satz-Extraktion mit Hilfe eines Dependency-Parsers ..... 28

3.3.2 Händische Durchsicht ..... 28

3.3.3 Entfernen von Abkürzungen ..... 29

3.3.4 Übersetzung ins Englische ..... 30

3.4 Fehlerinjektion ..... 31

3.5 Evaluationsdatensatz ..... 32

**4 Verwendete Modelle ..... 33**

4.1 Grammatical Error Correction: Tag, Not Rewrite ..... 33

4.1.1 Aufbau des Modells ..... 33

4.1.2 Vorverarbeitung des Datensatzes ..... 35

4.1.3 Verwendete Parameter ..... 36

4.1.4 Ensemblebildung ..... 37

4.2 Parallel Iterative Edit Models for Local Sequence Transduction ..... 37

4.2.1 Aufbau des Modells ..... 37

4.2.2 Vorverarbeitung der Daten ..... 37

4.2.3 Verwendete Parameter ..... 38

4.3 Fairseq-GEC ..... 39

4.3.1 Aufbau des Modells ..... 40

4.3.2 Vorverarbeitung der Daten ..... 41

4.3.3 Verwendete Parameter ..... 41

**5 Erste Evaluation der Ergebnisse und Vergleich mit den ursprünglichen Modellen ..... 43**

5.1 Verwendete Metriken ... 43
5.1.1 MaxMatch-Score ... 43
5.1.2 GLEU-Score ... 45
5.1.3 Sacre Bilingual Evaluation Understudy Score ... 45
5.1.4 Recall-Oriented Understudy for Gisting Evaluation ... 46
5.2 Einordnung der Ergebnisse ... 47
5.2.1 Vergleich zwischen den vortrainierten mit den entsprechenden nachtrainierten Modellen ... 47
5.3 Auswahl der erfolgreichsten Modelle für weiteres Retraining ... 60
**6 Finale Evaluation ... 61**
6.1 Vorstellung der Ergebnisse ... 61
6.2 Ergebnisse der Ensemblebildung der GECToR-Modelle und Vergleich mit den Bestwerten der einzelnen Modelle ... 63
6.3 Schlussfolgerungen und Interpretation ... 65
**7 Zusammenfassung und Ausblick ... 66**
**Literaturverzeichnis ... 68**
**Erklärung zur Urheberschaft ... 71**

# Abbildungsverzeichnis

Abbildung 1: Gegenüberstellung eines Satzpaares bestehend aus einem fehlerhaften und einem korrekten Satz. Die Fehler des linken Satzes sind farblich hervorgehoben. (https://www.grammarly.com/blog/engineering/adversarial-grammatical-error-correction/, 18.04.2021) ..... 10
Abbildung 2: Grundlegender Aufbau eines Encoder-Decoder-Seq2Seq-Modells. Die Encoder-Elemente sind blau und die Decoder-Bestandteile rot dargestellt. (Luong et al., 2015) ..... 12
Abbildung 3: Aufbau des GNMT-Modells. Dargestellt ist auch die Verteilung der Elemente über die GPUs während des Trainings. (Wu et al., 2016) ..... 13
Abbildung 4: Residual Connections visualisiert durch gebogene Pfeile beim überspringen von LSTM-Zellen. (Wu et al., 2016) ..... 14
Abbildung 5: RNMT+ Modellarchitektur. (Chen et al., 2018) ..... 15
Abbildung 6: Performance von RNMT+ im Vergleich zu anderen NMT-Modellen. (Chen et al., 2018) ..... 15
Abbildung 7: Visualisierung eines globalen Attention-Mechanismus. (Luong et al., 2015) ..... 16
Abbildung 8: Visualisierung eines lokalen Attention-Mechanismus. (Luong et al., 2015) ..... 16
Abbildung 9: Visualisierung des Input-Feeding-Ansatzes, der dazu verwendet wird, Alignment-Entscheidungen zu verbessern. (Luong et al., 2015) ..... 17
Abbildung 10: Evaluation der Modellkombinationen (Luong et al., 2015) ..... 17
Abbildung 11: Sprachbestandteile, die durch Part-of-Speech-Tagging erfasst werden. (http://partofspeech.org/, 18.04.2021) ..... 18
Abbildung 12: Veranschaulichung der Modellarchitekturen oben genannter Modelle. (Rei & Yannakoudakis, 2016) ..... 19
Abbildung 13: Evaluationsergebnisse der verschiedenen Modelle und Vergleich mit der Performance eines CRF. (Rei & Yannakoudakis, 2016) ..... 20
Abbildung 14: Evaluationsergebnisse zwischen den Ansätzen auf Wortebene und denen auf Char-Ebene. (Rei et al., 2016) ..... 21
Abbildung 15: Evaluationsergebnisse des LASERTAGGERS und Vergleich mit den Ergebnissen verschiedener GEC-Systeme. (Malmi et al., 2019) ..... 22

Abbildung 16: Veranschauchlichung der Modellarchitektur des CNN-basierten Ansatzes für GEC. (Chollampatt & Ng, 2018) ..... 23
Abbildung 17: Modellarchitektur des Modells, das auf einem hybriden Ansatz von Betrachtung auf Wort- und Zeichenebene aufbaut. (Ji et al., 2017) ..... 24
Abbildung 18: Ergebnisse der Blindtests zu Vergleich von DeepL mit anderen Übersetzungssystemen. (https://www.deepl.com/press.html, 18.04.2021) ..... 30
Abbildung 19: Aufbau eines Transformer-Encoders am Beispiel BERT (Vaswani et al., 2017) ..... 34
Abbildung 20: Aufbau des Fairseq-GEC-Modells zur Veranschaulichung des Kopiermechanismus von korrekten Tokens (Zhao et al., 2019) ..... 40
Abbildung 21: Diagramm zum Vergleich des ursprünglichen BERT-Modells mit dem nachtrainierten Modell über alle verwendeten Metriken. (eigene Abbildung) ..... 49
Abbildung 22: Diagramm zum Vergleich des ursprünglichen RoBERTa-Modells mit dem nachtrainierten Modell über alle verwendeten Metriken. (eigene Abbildung) ..... 52
Abbildung 23: Diagramm zum Vergleich des ursprünglichen XLNet-Modells mit dem nachtrainierten Modell über alle verwendeten Metriken. (eigene Abbildung) ..... 54
Abbildung 24: Diagramm zum Vergleich des ursprünglichen PIE-Modells mit dem nachtrainierten Modell über alle verwendeten Metriken. (eigene Abbildung) ..... 57
Abbildung 25: Diagramm zum Vergleich des ursprünglichen Fairseq-GEC-Modells mit dem nachtrainierten Modell über alle verwendeten Metriken. (eigene Abbildung) ..... 59
Abbildung 26: Vergleich aller vollständig nachtrainierten Modelle untereinander zur Bestimmung des erfolgreichsten Modells. (eigene Abbildung) ..... 63
Abbildung 27: Vergleich der gebildeten Ensembles aus den vollständig nachtrainierten GECToR-Modellen mit dem erfolgreichsten einzelnen Modell. (eigene Abbildung) .... 65

# Tabellenverzeichnis

Tabelle 1: Gegenüberstellung von fehlerhaften und korrekten Sätzen nach der Fehlerinjektion. ..... 32
Tabelle 2: Veranschaulichung des Ergebnisses der Vorverarbeitung für die GECToR-Modelle. ..... 36
Tabelle 3: Veranschauchlichung des Ergebnisses der Vorverarbeitung für das Modell PIE. ..... 38
Tabelle 4: Veranschauchlichung des Ergebnisses der Vorverarbeitung für das Modell Fairseq-GEC. ..... 41
Tabelle 5: Veranschauchlichung der Source-Gold-Annotation, die für den MaxMatch-Score verwendet wird. ..... 44
Tabelle 6: Vergleich des nachtrainierten GECToR-BERT-Modells mit dem ursprünglichen Modell mit Hilfe des M²-Scores. ..... 48
Tabelle 7: Vergleich des nachtrainierten GECToR-BERT-Modells mit dem ursprünglichen Modell mit Hilfe des GLEU-Scores. ..... 48
Tabelle 8: Vergleich des nachtrainierten GECToR-BERT-Modells mit dem ursprünglichen Modell mit Hilfe des SacreBLEU-Scores. ..... 49
Tabelle 9: Vergleich des nachtrainierten GECToR-BERT-Modells mit dem ursprünglichen Modell mit Hilfe des ROUGE-L-Scores. ..... 49
Tabelle 10: Vergleich des nachtrainierten GECToR-RoBERTa-Modells mit dem ursprünglichen Modell mit Hilfe des M²-Scores. ..... 50
Tabelle 11: Vergleich des nachtrainierten GECToR-RoBERTa-Modells mit dem ursprünglichen Modell mit Hilfe des GLEU-Scores. ..... 51
Tabelle 12: Vergleich des nachtrainierten GECToR-RoBERTa-Modells mit dem ursprünglichen Modell mit Hilfe des SacreBLEU-Scores. ..... 51
Tabelle 13: Vergleich des nachtrainierten GECToR-XLNet-Modells mit dem ursprünglichen Modell mit Hilfe des M²-Scores. ..... 53
Tabelle 14: Vergleich des nachtrainierten GECToR-XLNet-Modells mit dem ursprünglichen Modell mit Hilfe des GLEU-Scores. ..... 53
Tabelle 15: Vergleich des nachtrainierten GECToR-XLNet-Modells mit dem ursprünglichen Modell mit Hilfe des SacreBLEU-Scores. ..... 53

Tabelle 16: Vergleich des nachtrainierten GECToR-XLNet-Modells mit dem ursprünglichen Modell mit Hilfe des ROUGE-L-Scores. ..... 54
Tabelle 17: Vergleich der eigenen GECToR-Modelle mit dem ursprünglichen Modell mit Hilfe des M²-Scores. ..... 55
Tabelle 18: Vergleich der eigenen GECToR-Modelle mit dem ursprünglichen Modell mit Hilfe von GLEU, SacreBLEU und ROUGE-L. ..... 56
Tabelle 19: Vergleich des nachtrainierten PIE-Modells mit dem ursprünglichen Modell mit Hilfe des M²-Scores. ..... 56
Tabelle 20: Vergleich des nachtrainierten PIE-Modells mit dem ursprünglichen Modell mit Hilfe des GLEU- und SacreBLEU-Scores. ..... 56
Tabelle 21: Vergleich des nachtrainierten PIE-Modells mit dem ursprünglichen Modell mit Hilfe des ROUGE-L-Scores. ..... 57
Tabelle 22: Vergleich des nachtrainierten Fairseq-GEC-Modells mit dem ursprünglichen Modell mit Hilfe des M²-Scores. ..... 58
Tabelle 23: Vergleich des nachtrainierten Fairseq-GEC-Modells mit dem ursprünglichen Modell mit Hilfe des GLEU- und SacreBLEU-Scores. ..... 58
Tabelle 24: Vergleich des nachtrainierten Fairseq-GEC-Modells mit dem ursprünglichen Modell mit Hilfe des ROUGE-L-Scores. ..... 59
Tabelle 25: Vergleich der verschiedenen Modellarchitekturen anhand des besten Modells aus 20 Epochen Nachtrainieren. Evaluation mittels M²-Score. ..... 61
Tabelle 26: Vergleich der verschiedenen Modellarchitekturen anhand des besten Modells aus 20 Epochen Nachtrainieren. Evaluation mittels GLEU-Score. ..... 62
Tabelle 27: Vergleich der verschiedenen Modellarchitekturen anhand des besten Modells aus 20 Epochen Nachtrainieren. Evaluation mittels SacreBLEU-Score. ..... 62
Tabelle 28: Vergleich der verschiedenen Modellarchitekturen anhand des besten Modells aus 20 Epochen Nachtrainieren. Evaluation mittels ROUGE-L-Score. ..... 62
Tabelle 29: Vorstellung der Ergebnisse der Ensemblebildung der GECToR-Modelle gemessen am M²-Score. ..... 63
Tabelle 30: Vorstellung der Ergebnisse der Ensemblebildung der GECToR-Modelle gemessen am GLEU- und SacreBLEU-Score. ..... 64
Tabelle 31: Vorstellung der Ergebnisse der Ensemblebildung der GECToR-Modelle gemessen am ROUGE-L-Score. ..... 64

# Codeverzeichnis

Code 1: Extraktion von Einträgen aus dem Ticketsystem der Firma HORSCH mittels eines Webcrawlers. Benutzt wurde das Chromedriver-Framework. ..... 27

Code 2: Dependency-Parsing. Filtern der analysierten Einträge nach der Wurzel (ROOT), sowie Überprüfung ob es sich um ein Verb handelt..... 28

Code 3: Dynamische Entfernung von Abkürzungen. Erkennt der Algorithmus eine unbekannte Abkürzung wird der Benutzer aufgefordert eine ausgeschriebene Form einzugeben. ..... 29

Code 4: Übersetzung der Sätze des Datensatzes mittels der API von DeepL. ..... 31

Code 5: Überführen von Ausgangs- und Zielsätzen in die Source-Gold-Annotation. .. 44

## Zusammenfassung

## Abstract

# 1 Einleitung

In Zeiten der Globalisierung und dem weltweiten Export von Maschinen stellt sich die Firma HORSCH die Frage, wie sich die Firma, Händler und Kunden untereinander effizient vernetzen und miteinander kommunizieren können. Diese Frage führte zur Idee der Einführung einer Plattform, auf der Kunden des Konzerns Fragen und Probleme sowohl an den Betrieb direkt, als auch an Händler und andere Kunden stellen können, um Lösungen oder Anregungen weitergeben zu können.

Problem an diesem Vorhaben ist jedoch, dass es Schwierigkeiten bei der Kommunikation aufgrund sprachlicher Differenzen und Barrieren geben könnte. Zwar ist Englisch als Weltsprache der Ausgangspunkt dieses Informationsaustauschs, jedoch kann es aus verschiedensten Gründen zu erheblichen Unterschieden der Sprachkompetenzen zwischen den Kommunikationspartner kommen. Überlegungen diese Komplikation zumindest teilweise zu lösen, führte zum Themengebiet der Grammatical Error Correction. Dieses Feld beschäftigt sich mit Systemen, die potentiell grammatikalisch inkorrekte Sätze/Sequenzen in eine korrekte Form überführt. Damit soll der Austausch zwischen den Parteien eines Dialogs erleichtert werden.

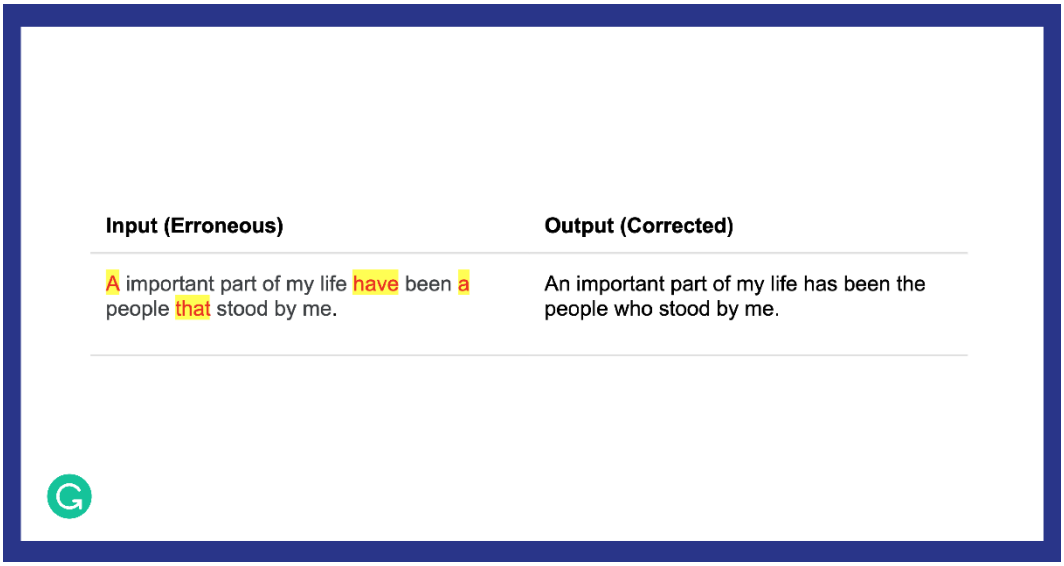


**Abbildung 1: Gegenüberstellung eines Satzpaares bestehend aus einem fehlerhaften und einem korrekten Satz. Die Fehler des linken Satzes sind farblich hervorgehoben. (https://www.grammarly.com/blog/engineering/adversarial-grammatical-error-correction/, 18.04.2021)**

Grammatical Error Correction gewann über die Zeit immer mehr an Bedeutung und ist heute ein bedeutender Task im Feld des Natural Language Processing. Die Ursprünge

der automatischen, maschinellen Fehlerkorrektur gehen auf Ansätze zurück, die fehlerspezifische Machine-Learning-Classifier aufbauen und trainieren. Mit der Zeit wandelte sich der Grundgedanke der Systeme, hin zu einer einsprachigen Übersetzungsaufgabe. Dabei soll ein System einen gegebenen fehlerhaften Text in seine korrekte Form „übersetzen". Mit steigendem Fortschritt im Bereich der neuronalen Netze wurden statistische Ansätze mit Systemen kombiniert, die auf neuronalen Netzen basieren, oder vollständig durch Deep Learning Ansätze verdrängt. (Raheja & Alikaniotis, 2020) Ein weiterer Ansatz diesem Problem der Fehlerkorrektur zu begegnen, ist es, Fehler zu annotieren und anschließend anhand verschiedener Tags auszubessern. (Omelianchuk et al., 2020)
Mit dieser Arbeit wird der Versuch unternommen, bestehende Grammatical Error Correction Systeme nachzutrainieren und anschließend systematisch zu vergleichen. Dazu wurden Arbeiten ausgewählt, die dem aktuellen State-of-the-Art entsprechen und dessen Code, als auch vortrainierte Modelle, öffentlich zugänglich sind.

# 2 Related Work

Im folgenden Kapitel werden Arbeiten und Techniken vorgestellt, die in vergangenen Werken verwendet wurden und hilfreich für die Umsetzung dieser Arbeit sein können. Zu Beginn soll das Thema Neural Machine Translation (NMT) definiert, sowie näher beleuchtet werden. Anschließend wird auf das Feld Sequence Tagging näher eingegangen. Zuletzt werden Arbeiten aus der Grammatical Error Correction (GEC) vorgestellt, die aufgrund verschiedender Faktoren nicht in dieser Arbeit verwendet wurden.

## 2.1 Neural Machine Translation

Neural Machine Translation hat die Zielsetzung eine Eingangssequenz in eine Sequenz von Wörtern einer anderen Sprache zu überführen. Gängig ist es hierbei Encoder-Decoder-Netze aufzubauen. Der Encoder überführt die einzelnen Tokens einer Sequenz in eine Vektorrepräsentation und bildet den State-Vektor. Die Aufgabe des Decoders ist es anschließend jeden Token des Zielsatzes anhand des State-Vektors vorherzusagen. (DeepAI, 2020)

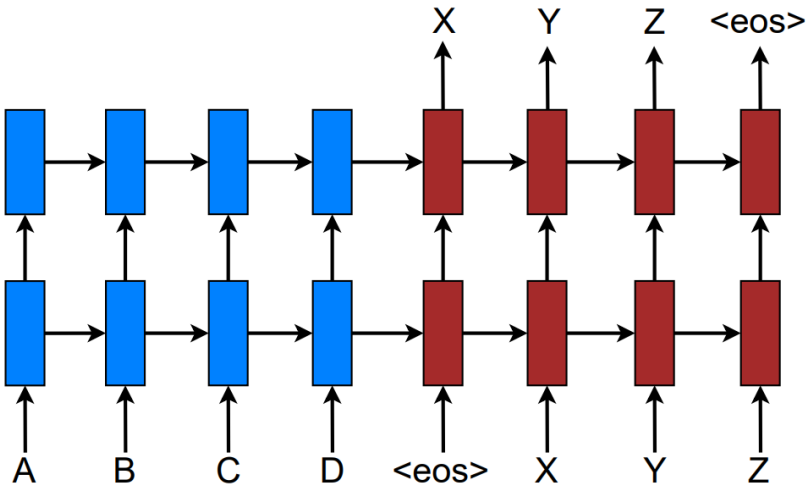


**Abbildung 2: Grundlegender Aufbau eines Encoder-Decoder-Seq2Seq-Modells. Die Encoder-Elemente sind blau und die Decoder-Bestandteile rot dargestellt. (Luong et al., 2015)**

Die klassische Encoder-Decoder-Architektur kann durch weitere Layer, wie z.B. Attention-Mechanismen oder Residual-Connections, ergänzt werden. Diese Erweiterungen können dabei helfen, die Performance beim Umgang mit komplexen Sätzen o.ä. zu verbessern. Deshalb werden im folgenden verschiedene Ansätze und Erweiterung dieser Grundarchitektur vorgestellt. Die Relevanz dieses Themas für meine Arbeit ergibt sich aus dem Ansatz GEC-Systeme als Encoder-Decoder-Modelle aufzubauen, da man

GEC auch als Übersetzung von Learner Language in eine korrekte Sprache betrachten kann.

Google's Neural Machine Tranlsation (GNMT) ist ein gutes Beispiel für ein Encoder-Decoder Sequence-2-Sequence-Modell. Es besteht aus acht Encoder und acht Decoder-Modellen. Das Encoder-Netzwerk produziert eine Liste von Vektoren. Jeder Vektor steht dabei für einen Token. Das Decoder-Netz in Kombination mit einem abschließenden Softmax-Layer kann nun Token für Token eine Sequenz aus dieser Liste von Vektoren aufbauen. Dies wird solange gemacht, bis das End-of-Sentence-Symbol vorhergesagt wurde. Zusätzlich sind Encoder und Decoder durch ein Attention-Netzwerk miteinander verbunden. Der Attention-Mechanismus erlaubt es dem Decoder sich während des Vorgangs auf verschiedene Teile des Ausgangssatzes zu fokussieren. Herausstechend ist ebenfalls, dass die erste Schicht des Encoders bidirektional ist, wobei alle anderen Layer des Encoders, sowie Decoders unidirektional sind. Diese Bidirektionalität wird dazu verwendet, um den Kontext der Input-Sequenz besser erfassen zu können. Um die Parallelisierung während der Berechnung weitgehend aufrecht erhalten zu können, wurde bei den anderen Schichten auf Unidirektionalität gesetzt.

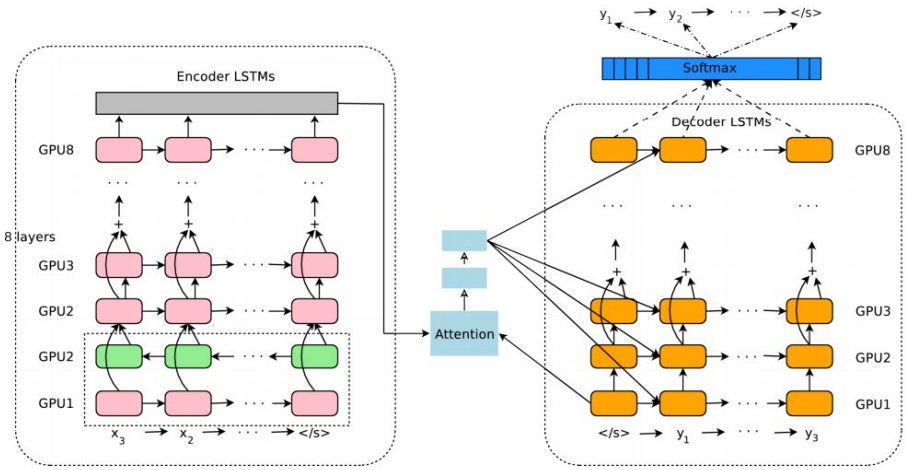


**Abbildung 3: Aufbau des GNMT-Modells. Dargestellt ist auch die Verteilung der Elemente über die GPUs während des Trainings. (Wu et al., 2016)**

Eine weitere Besonderheit dieses Modells ist die Verwendung einer Mischung aus Vorhersagen auf Wort- bzw. Buchstabenebene. Das Modell benutzt ein festes Vokabular, um Annahmen auf Wortebene zu tätigen. Stößt das Modell jedoch auf unbekannte Wörter kann es auf Vorhersagen auf Buchstabenebene zurückgreifen, um ein passendes Wort zu generieren.

Bei der Evaluation griffen die Entwickler unter anderem auf die Bewertung der Übersetzungen durch Menschen zurück. Dabei wurde ein früheres, statistisches Übersetzungsmodell, dieses Modell und die Übersetzung durch Menschen miteinander verglichen. Dabei konnte sich das Modell dieser Arbeit durchschnittlich um mehr als 60% gegenüber dem statistischen Modell verbessern. (Wu et al., 2016)

Ein weiteres Modell ist RNMT+ aus der Arbeit The Best of Both Worlds: Combining Recent Advances in Neural Machine Translation von Chen et al. Aufbauend auf GNMT werden bei RNMT+ mehrere Encoder und Decoder Schichten verwendet. Der Encoder dieses Modells besteht aus sechs bidirektionalen LSTM-Schichten, gefolgt von sieben unidirektionalen LSTMs. Bevor die Ergebnisse der bidirektionalen Schichten an die nächste Schicht weitergegeben werden, werden sie verkettet. Der Decoder besteht wie bei GNMT aus acht unidirektionalen LSTM-Schichten. Ab der dritten Schicht des Encoders, sowie Decoders, wurden Residual Connections angefügt. Residual Connections helfen dabei, tiefere Netzwerke zu trainieren, da man dem vanishing gradient problem entgegenwirken kann, bei dem der Gradient gegen null strebt und Änderungen in den tiefen Schichten eines Netzes stark verlangsamt werden. (Gesellschaft für Informatik e.V., 2017)

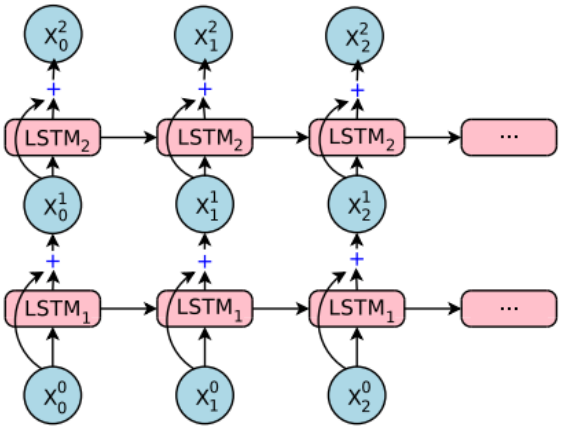


**Abbildung 4: Residual Connections visualisiert durch gebogene Pfeile beim überspringen von LSTM-Zellen. (Wu et al., 2016)**

Des Weiteren wurde in jede LSTM-Zelle ein Normalisierungsmechanismus verankert, der durch die Transformer-Modell inspiriert wurde. Die Normalisierung hilft dabei, das Training zu stabilisieren. Zum Output des Encoders wurde ein projection layer hinzugefügt, der das Ziel hat, die Komplexität der Vektorrepräsentation der Daten des Encoders zu reduzieren und mit der erwarteten Form des Inputs des Decoders

abzugleichen. RNMT+ setzt, genau wie GNMT, auf einen Attention-Mechanismus. Anders als bei GNMT findet hier jedoch ein Multihead-Attention-Layer Verwendung.

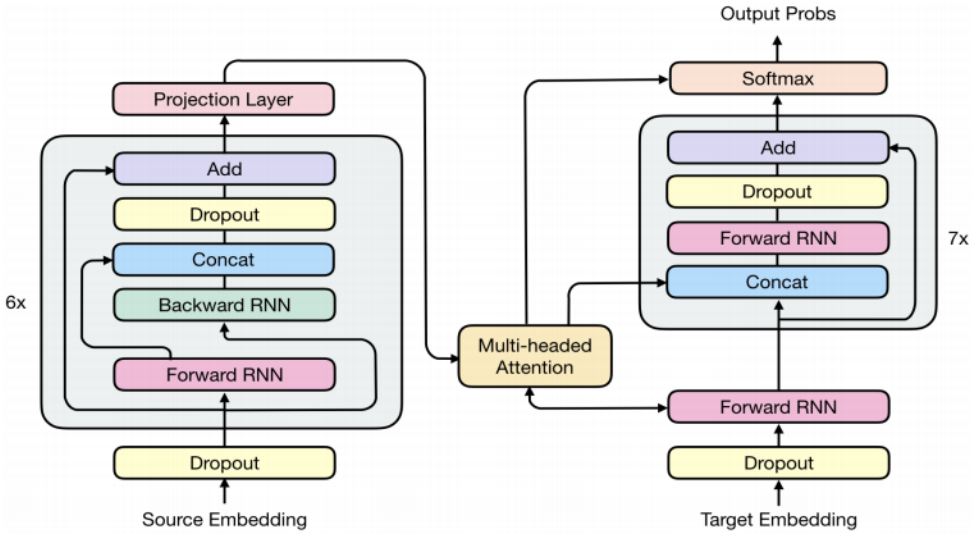


**Abbildung 5: RNMT+ Modellarchitektur. (Chen et al., 2018)**

Mit diesem Modell konnte auf dem WMT14 En→FR Datensatz unter fünf Modellen das beste Ergebnis erreicht werden. Beim Vergleich wurden neben dem Modell von Chen et al. GNMT, ein Convolutional Sequence-2-Sequence-Modell und zwei Transformer-Modelle verwendet. (Chen et al., 2018)

| Model | Test BLEU | Epochs | Training Time |
|---|---|---|---|
| GNMT | 38.95 | - | - |
| ConvS2S [7] | $39.49 \pm 0.11$ | 62.2 | 438h |
| Trans. Base | $39.43 \pm 0.17$ | 20.7 | 90h |
| Trans. Big [8] | $40.73 \pm 0.19$ | 8.3 | 120h |
| RNMT+ | $41.00 \pm 0.05$ | 8.5 | 120h |

**Abbildung 6: Performance von RNMT+ im Vergleich zu anderen NMT-Modellen. (Chen et al., 2018)**

In der Arbeit Effective Approaches to Attention-based Neural Maschine Translation untersuchten Luong et al. den Einfluss von verschiedenen Attention-Mechanismen auf die Performance von NMT-Systemen. Dabei betrachteten sie speziell lokale und globale Attention. Wird die Attention auf alle Positionen der Source-Tokens platziert, spricht man von globaler Attention. Lokale Attention dagegen betrachtet lediglich ein paar Tokens einer Sequenz. Ziel aller Attention-Schichten ist es, einen Kontext-Vektor aus dem State-Vektor zu bilden, der kontextuelle Informationen der Source-Tokens beinhaltet.

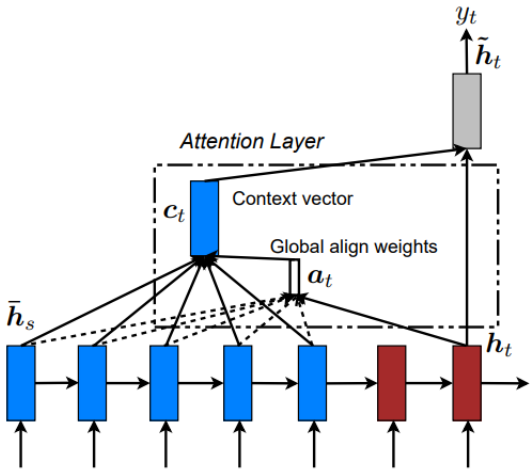


**Abbildung 7: Visualisierung eines globalen Attention-Mechanismus. (Luong et al., 2015)**

Bei der globalen Attention werden alle State-Vektoren des Encoders bei der Berechnung des Kontext-Vektors berücksichtigt. Die Berechnung erfolgt aus dem gewichteten Durchschnitt gemäß dem Alignment-Vektor über alle State-Vektoren.

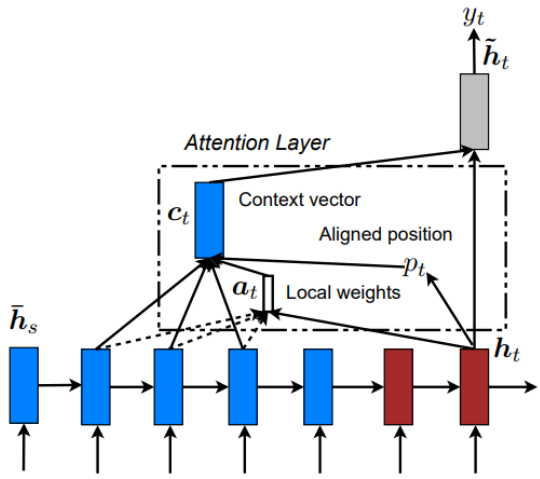


**Abbildung 8: Visualisierung eines lokalen Attention-Mechanismus. (Luong et al., 2015)**

Bei der lokalen Attention wird zuerst eine ausgerichtete Position des Zielwortes berechnet. Anschließend um die Position des Source-Tokens ein Fenster zentriert. Der Kontextvektor ergibt sich dann aus dem gewichteten Durchschnitt der State-Vektoren in diesem Fenster.

Um bei zukünftigen Entscheidungen über die Ausrichtung zwischen Tokens des Source- und Zielsatzes vergangene Entscheidungen zu berücksichtigen, beschäftigten sich Luong et al. mit ihrem eigenen Input-Feeding-Ansatz. Bei diesem Ansatz werden Attentionvektoren mit den Inputs beim nächsten Zeitschritt verkettet. Davon erhofften sie

sich, dass sich das Modell über alle vergangenen Entscheidungen beim Alignment zwischen Source- und Zielsatz bewusst ist. Ein weiterer Vorteil dieses Ansatzes ist tiefe Ausbreitung des Netzwerks, sowohl horizontal, als auch vertikal.

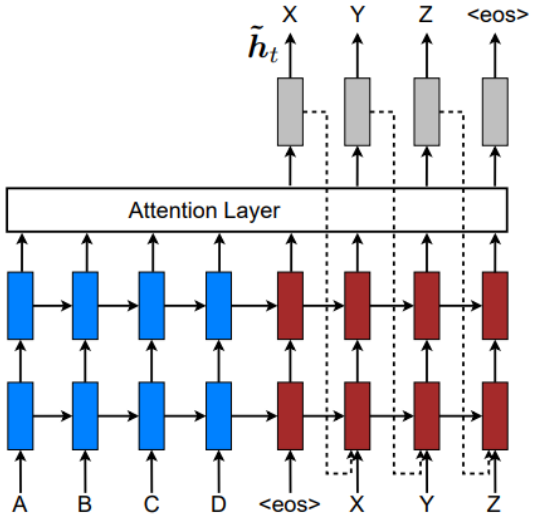


**Abbildung 9: Visualisierung des Input-Feeding-Ansatzes, der dazu verwendet wird, Alignment-Entscheidungen zu verbessern. (Luong et al., 2015)**

Die Autoren testeten verschiedene Kombinationen von Modellen und evaluierten es gegen ihr Basismodell. Dabei konnte festgestellt werden, dass die behandelten Mechanismen zur Verbesserung der Performance solcher Netzwerke beitragen können. (Luong et al., 2015)

| | | |
|---|---|---|
| *Our NMT systems* | | |
| Base | 10.6 | 11.3 |
| Base + reverse | 9.9 | 12.6 (*+1.3*) |
| Base + reverse + dropout | 8.1 | 14.0 (*+1.4*) |
| Base + reverse + dropout + global attention (*location*) | 7.3 | 16.8 (*+2.8*) |
| Base + reverse + dropout + global attention (*location*) + feed input | 6.4 | 18.1 (*+1.3*) |
| Base + reverse + dropout + local-p attention (*general*) + feed input | 5.9 | 19.0 (*+0.9*) |
| Base + reverse + dropout + local-p attention (*general*) + feed input + unk replace | | 20.9 (*+1.9*) |
| *Ensemble* 8 models + unk replace | | **23.0** (*+2.1*) |

**Abbildung 10: Evaluation der Modellkombinationen (Luong et al., 2015)**

## 2.2 Sequence Tagging

Als Sequence Tagging oder Sequence Labeling bezeichnet man im Feld des Natural Language Processing (NLP) eine Aufgabe, bei dem jedem Token in einer Sequenz eine Annotation aus einem Satz vordefinierter Annotationen zugewiesen werden soll. Es gibt viele gängige Tasks für Sequence Tagging. Das wohl bekannteste Beispiel ist das part-of-Speech-Tagging (POS-Tagging), bei dem jedem Wort in einer Eingangssequenz eine Wortart zugeordnet wird. (Medium, 2020)


**Abbildung 11: Sprachbestandteile, die durch Part-of-Speech-Tagging erfasst werden. (http://partofspeech.org/, 18.04.2021)**

Weitere Anwendungsfälle von Sequence Labeling sind beispielsweise Named Entity Recognition oder Metaphererkennung. Relevant für diese Arbeit ist das Labeln von Fehlern in einem Text (Error Tagging). In diesem Gebiet geht es darum, grammatikalische Fehler in einem Satz oder einer Sequenz zu erkennen und zu annotieren. Einige GEC-Modelle bauen auf dieser Technik der Fehlererkennung auf, um diese anschließend weiterzuverarbeiten.

In der Arbeit Compositional Sequence Labeling Models for Error Detection in Learner Writing präsentieren Rei und Yannakoudakis einen systematischen Vergleich von sechs verschiedenen Herangehensweisen ein Modell zur Fehlererkennung aufzubauen. Dabei handelt es sich speziell um folgende Architekturen:

- Convolutional Neural Network (CNN) (a)
- Deep-CNN (b)
- Bidirektionales Recurrent Neural Network (Bi-RNN) (c)
- Deep Bi-RNN (d)
- Bidirektionales Long-Short-Term-Memory-Network (Bi-LSTM) (c)
- Deep Bi-LSTM (d)

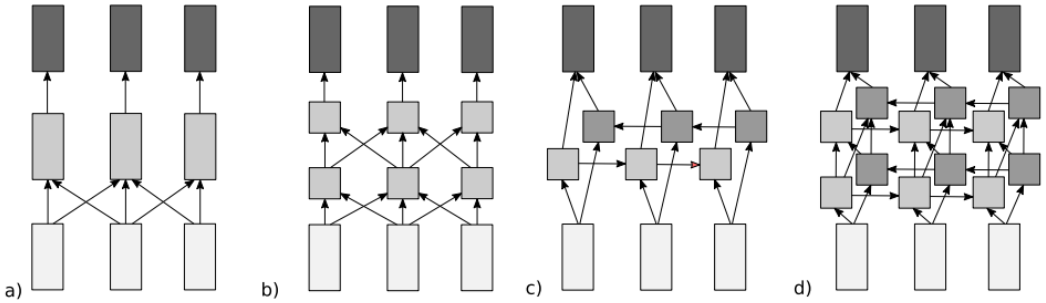


**Abbildung 12: Veranschaulichung der Modellarchitekturen oben genannter Modelle. (Rei & Yannakoudakis, 2016)**

Die Startschicht jedes Models ist ein Word-Embedding, bei dem jeder Token in eine Vektorrepräsentation überführt wird. In der Mitte sind die jeweiligen Komponenten der Architekturen zu finden. Den Abschluss jedes dieser Netzwerke bildet eine Softmax-Schicht, um eine Wahrscheinlichkeitsverteilung über alle möglichen Annotationen zu erhalten.

Beim CNN wird wird der hidden vector auf Basis eines festen Kontextfensters berechnet. Da das CNN als Feed-Forward-Netzwerk fungiert, ist das Ziel dieser Architektur das Erkennen von verschiedenen n-gram-Typen in einer Sequenz. Beispielsweise könnte das Netz somit falsche Wortfolgen aus den Trainingsdaten erkennen, sich einprägen und anschließend in unbekannten Daten wiedererkennen. Beim Deep-CNN basiert die Fehlererkennung auf demselben Prinzip. Es kommt jedoch durch die tiefere Architektur hinzu, dass komplexere Features (n-grams) vom Datensatz besser erfasst werden können.

Ein RNN-Netzwerk basiert auf der Berechnung des State-Vektors auf Basis des aktuellen Tokens in Kombination mit dem State-Vektor der letzten Iteration. Die Bidirektionalität entsteht durch zwei RNN-Zellen. Eine der beiden Zellen durchläuft einen Satz von links nach rechts, die andere Zelle arbeitet von rechts nach links. Die beiden State-Vektoren können anschließend verbunden werden und als neuer State-Vektor fungieren. Für die Error-Detection soll ein Netzwerk dieser Architektur semantische Anomalien oder ungrammatikalische Kombinationen erlernen. Wie auch beim Deep-CNN soll das Deep-RNN in der Lage sein, komplexere Features aus dem Datensatz zu extrahieren.

LSTM-Netzwerke nutzen zwei separate Vektoren, um Informationen zwischen den Iterationen zu verarbeiten. Außerdem verwenden sie einen Gating-Mechanismus, um ihren Output zu modulieren. Die Bidirektionalität funktioniert analog zu der des RNN-

Netzwerks. Ein Vorteil des LSTMs ist, dass es durch den Gating-Mechanismus lernt, welche Operationen in einer Iteration erforderlich sind und somit high-level Features extrahieren kann. Um noch komplexere Features zu betrachten, wurde ebenso mit einem Deep Bi-LSTM experimentiert.

Für die Evaluation wurden oben genannte Modelle mit einem Conditional Random Field (CRF) verglichen. Ein CRF ist ein statistisches Modell, das oft im Bereich des Sequence Taggings verwendet wird. Die Experimente zeigen, dass jedes neuronale Netz besser abschneiden konnte als das CRF. Die besten Ergebnisse lieferten Bi-RNN und Bi-LSTM. (Rei & Yannakoudakis, 2016)

| | Development | | | Test | | | | |
|---|---|---|---|---|---|---|---|---|
| | P | R | $F_{0.5}$ | predicted | correct | P | R | $F_{0.5}$ |
| CRF | 62.2 | 13.6 | 36.3 | 914 | 516 | 56.5 | 8.2 | 25.9 |
| CNN | 52.4 | 24.9 | 42.9 | 3518 | 1620 | 46.0 | 25.7 | 39.8 |
| Deep CNN | 48.4 | 26.2 | 41.4 | 3992 | 1651 | 41.4 | 26.2 | 37.1 |
| Bi-RNN | **63.9** | 18.0 | 42.3 | 2333 | 1196 | **51.3** | 19.0 | 38.2 |
| Deep Bi-RNN | 60.3 | 17.6 | 40.6 | 2543 | 1255 | 49.4 | 19.9 | 38.1 |
| Bi-LSTM | 54.5 | **28.2** | **46.0** | 3898 | 1798 | 46.1 | **28.5** | **41.1** |
| Deep Bi-LSTM | 56.7 | 21.3 | 42.5 | 2822 | 1359 | 48.2 | 21.6 | 38.6 |

**Abbildung 13: Evaluationsergebnisse der verschiedenen Modelle und Vergleich mit der Performance eines CRF. (Rei & Yannakoudakis, 2016)**

Rei und Sogard Beschäftigen sich in ihrer Arbeit ebenfalls teilweise mit dem Error Tagging Task. Ihr Modell baut auf einem bidirektionalen LSTM-Netzwerk auf. Ergänzt wird die Architektur durch einen dynamischen Attention-Mechanismus. Die Architektur des Modells erinnert stark an Modelle, die im Bereich der NMT zu finden sind. Die Tokens werden zuerst in einer Sequenz an Wortrepräsentationen abgebildet. Diese Wortrepräsentationen setzen sich aus einer Embedding-basierten sowie Character-basierten zusammen. Sie werden dem LSTM-Netz übergeben. Das Netzwerk durchläuft einen Satz in beide Richtungen. Daraus entsteht für jeden Token ein State-Vektor, der den Kontext eines Tokens berücksichtigt. Anschließend kommt der Attention-Mechanismus zu tragen, der dynamisch darüber entscheiden kann, wie stark ein Wort zur Gesamtrepräsentation beiträgt. Das Modell wurde im Bereich des Error Taggings auf Satzebene trainiert. Die Ausgabe ist eine binäre Annotation, ob ein Satz korrekt ist oder nicht. Außerdem beinhaltet die Ausgabe einen Confidence-Score, mit dem analysiert werden kann, mit welcher Sicherheit die Vorhersage des Systems getätigt wurde. Bei der Evaluation auf

dem FCE Testdatensatz, konnte das System mit einem F1-Score von 85,14 relativ gute Werte für die Fehlererkennung auf Satzebene erzielen. (Rei & Sogaard, 2018)

Rei et al. Entwickelten für ihre Arbeit Attending to Characters in Neural Sequence Labeling Models ein Modell, das neben Wortebene auch die Zeichenebene brücksichtigt. Dies geschieht vor allem bei seltenen oder dem system unbekannten Wörtern. Die Basis des Modells ist ein elementares neuronales Netz, wie es häufig im Bereich des Sequence Labeling zum Einsatz kommt. Beginnend mit dem Einbetten der Tokens in ein Word Embedding werden die Vektorrepräsentationen anschließend an ein bidirektionales LSTM weitergegeben. Bei der Verkettung beider State-Vektoren entsteht anschließend ein einzelner State-Vektor, der auf den gesamten Kontext konditioniert ist. Abschluss des Netzes auf Wortebene ist in diesem Fall keine Softmax-Schicht, sondern ein CRF, das mit den Vektoren der LSTM-Schichten arbeitet.

Für die Behandlung auf Zeichenebene werden zwei Ansätze betrachtet. Für den ersten Ansatz wird zunächst jeder Token in einzelne Zeichen zerlegt und in ein Character-Embedding überführt. Anschließend durchlaufen diese Vektoren ein bidirektionales LSTM. Zuletzt werden die State-Vektoren verkettet, und einer nicht-linearen Schicht übergeben. Diese Schicht bildet aus den State-Vektoren eine Wortrepräsentation. Der zweite Ansatz verfolgt grundlegend dieselben Prinzipien. Anstatt jedoch zwei verschiedene Feature-Sets zu benutzen, können die beiden Netze die gleichen Repräsentationen erlernen. Anschließend kann man das Netz selbst entscheiden lassen, wie es die Informationen für jedes Wort kombiniert.

Für die Fehlerannotation wurde das Modell so trainiert, dass es eine binäre Entscheidung trifft, ob ein Token korrekt oder inkorrekt ist. Die Evaluation dieses Tasks wurde auf dem FCEPUBLIC-Datensatz durchgeführt. In folgender Tabelle können die Ergebnisse der einzelnen Modelle entnommen werden. (Rei et al., 2016)

| | CoNLL00 | | CoNLL03 | | PTB-POS | | FCEPUBLIC | |
|---|---|---|---|---|---|---|---|---|
| | DEV | TEST | DEV | TEST | DEV | TEST | DEV | TEST |
| Word-based | 91.48 | 91.23 | 86.89 | 79.86 | 96.29 | 96.42 | 46.58 | 41.24 |
| Char concat | 92.57 | 92.35 | 89.81 | 83.37 | 97.20 | 97.22 | 46.44 | 41.27 |
| Char attention | **92.92** | **92.67** | **89.91** | **84.09** | **97.22** | **97.27** | **47.17** | **41.88** |

**Abbildung 14: Evaluationsergebnisse zwischen den Ansätzen auf Wortebene und denen auf Char-Ebene. (Rei et al., 2016)**

Ein weiteres Modell, das diese Aufgabe lösen kann, ist LASERTAGGER von Malmi et al. Bei Tagging dieses Modell handelt es sich nicht um binäre Entscheidungen zwischen zwei Tags. Bei den Basis-Tags handelt es sich um KEEP und DELETE. Diese signalisieren, ob ein Token in den Zielsatz übernommen werden soll oder nicht. Zu den beiden Tags kommt ein „added phrase" P, der vor den entsprechenden Token eingefügt werden soll. P kann jedoch auch leer sein. Die Anzahl verschiedener Tags beläuft sich also auf 2*(Größe des Vokabulars). Für spezielle Anwendungsfälle, wie z.B. Sentence Fusion, wurden zusätzliche Tags ergänzt.

Das Modell arbeitet auf Basis eines Encoder-Decoder-Netzwerks. Für den Encoder wurde das Transformer-Modell BERT verwendet, auf das im Kapitel 4 näher eingegangen wird. Dazu wurde ein vortrainiertes BERT-Modell initialisiert. Als Decoder fungiert ein einschichtiger Transformer, um die Zusammenhänge zwischen den Labels zu modellieren. Zum weiteren Experimentieren wurde jedoch auch mit einem Klassischen Feed-Forward-Decoder gearbeitet.

Die Evaluation findet in diesem Fall nicht in Feld des Error Taggings statt. Es wird im Feld der GEC evaluiert, da das System in der Lage ist, Fehler zu erkennen und anschließend zu korrigieren. Der LASERTAGGER schneidet im Vergleich zu einem elementaren BERT-basierten Sequence-2-Sequence-Modell sehr gut ab, kann jedoch nicht mit State-of-the-Art Modellen, wie dem von Grundkiewicz et al. mithalten. (Malmi et al., 2019)

| Model | $P$ | $R$ | $F_{0.5}$ |
|---|---|---|---|
| Grundkiewicz et al. (2019) | *70.19* | *47.99* | *64.24* |
| SEQ2SEQ$_{\text{BERT}}$ | 6.13 | 14.14 | 6.91 |
| LASERTAGGER$_{\text{FF}}$ | 44.17 | 24.00 | 37.82 |
| LASERTAGGER$_{\text{AR}}$ | **47.46** | **25.58** | **40.52** |

**Abbildung 15: Evaluationsergebnisse des LASERTAGGERS und Vergleich mit den Ergebnissen verschiedener GEC-Systeme. (Malmi et al., 2019)**

## 2.3 Grammatical Error Correction

Zuletzt sollen nun noch Arbeiten zum Thema Grammatical Error Correction vorgestellt werden, die nicht in dieser Arbeit verwendet wurden.

Chollampatt und Ng präsentieren in Ihrer Arbeit ein CNN-basiertes Modell für Grammatical Error Correction. Dieses Modell besteht wiederrum aus Encoder und Decoder. Nachdem eine Sequenz in das Word Embedding überführt wurde, folgen sieben

Encoder-Schichten, die aus Convolutional-Layers aufgebaut sind. Danach beginnt die Sequenzgenerierung mit Hilfe des Decoders. Dieser ist ebenfalls aus sieben Schichten mit Convolutional-Layers aufgebaut. Hinzu kommen jedoch sieben Attention-Module, wobei jedes dieser Module für eine Decoder-Schicht vorgesehen ist. Die letzte Schicht des Decoders ist eine Softmax-Schicht. Damit lässt sich eine Wahrscheinlichkeitsverteilung über das Vokabular bilden und den nächsten Token berechnen.

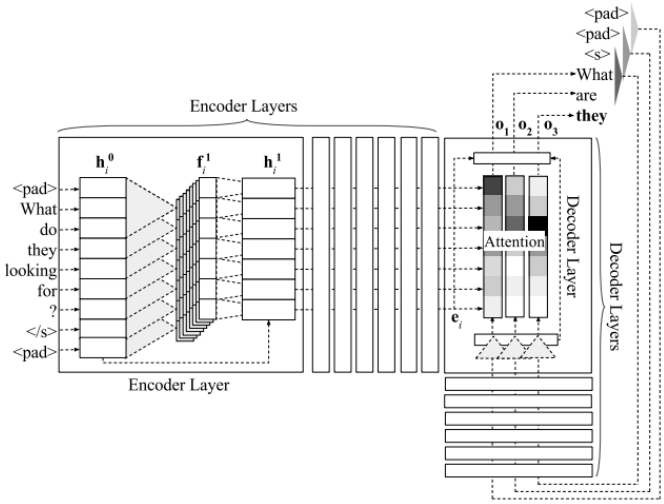


**Abbildung 16: Veranschauchlichung der Modellarchitektur des CNN-basierten Ansatzes für GEC. (Chollampatt & Ng, 2018)**

Die Evaluation fand auf dem CoNLL-2014 shared task Datensatz statt. Benutzt wurde der MaxMatch-Score, welcher Werte für Precision, Recall und F0.5-Score liefert. Bei dieser Evaluation konnte das CNN-Modell an den State-of-the-Art anknüpfen und erreichte einen F0.5 von 54,79. Eine zweite Evaluation auf dem JFLEG Testdatensatz mit der GLEU-Metrik zeigte, dass das vorgestellte System besser als alle dagewesenen Encoder-Decoder-Modelle abschneidet. (Chollampatt & Ng, 2018)

Ji et al. stellen in ihrer Arbeit A Nested Attetion Neural Hybrid Model for Grammatical Error Correction ein Modell vor, das mit unbekannten Wörtern besser umgehen soll. Die Basis ihres Modells ist ein Encoder-Decoder-Modell mit einem Attention-Mechanismus. Als Zusatz betteten sie in ihr Modell einen Encoder auf Zeichenebene ein, der es möglich machen soll, mit unbekannten Wörtern hinreichend umzugehen. Analog dazu wurde auch ein zusätzlicher Decoder eingebettet, der auf Zeichenebene arbeitet. Beide Komponenten der Character-Ebene arbeiten mit einem Attention-Mechanismus.

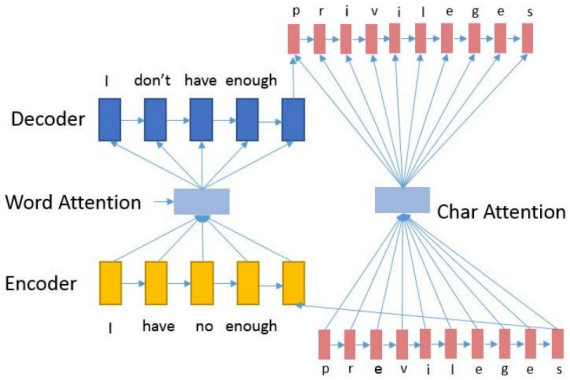


**Abbildung 17: Modellarchitektur des Modells, das auf einem hybriden Ansatz von Betrachtung auf Wort- und Zeichenebene aufbaut. (Ji et al., 2017)**

Zur Evaluation ihres Systems verglichen sie ihr Modell mit zwei anderen Architekturen. Das erste Vergleichssystem war ein NMT-Modell auf Wortebene, welches mit einem Attention-Mechanismus und Postprocessing für unbekannte Wörter arbeitet. Bei diesem Postprocessing-Schritt wird der wahrscheinlichste Token aus einem Korrekturlexikon für den unbekannten Token eingefügt. Als zweite Vergleichsbasis wird ein Modell verwendet, das bereits hybrid (Wort- und Char-Level) arbeitet, jedoch nur ein Level an Attention implementiert. Bei diesem Vergleich Schnitt das Modell von Ji et al. besser als die genannten Vergleichssysteme ab und erreichte einen F0.5 von 41,53. (Ji et al., 2017)

# 3 Datenerhebung

In diesem Kapitel wird die Erhebung der Daten zur Erstellung eines geeigneten Datensatzes im Anwendungskontext der Firma HORSCH beschrieben. Landwirtschaftliches Vokabular unterscheidet sich stark vom Wortschatz, den man in bereits existierenden Datensätzen vorfindet. Deswegen ist es unerlässlich, einen eigenen Datensatz in das Training der Modelle einfließen zu lassen. Die verwendeten Daten wurden aus einem Ticketsystem für Händler der Firma HORSCH extrahiert und anschließend weiterverarbeitet, um sicherzustellen, dass nur ganze Sätze in das System fließen. Des Weiteren wurden alle Abkürzungen aus dem Datensatz entfernt und dieser in Folge dessen ins Englische übersetzt. Da die Größe des Datensatzes von HORSCH nicht ausreichen würde, um Modelle erfolgreich nachtrainieren zu können, wurde dieser mit einem weiteren Datensatz, bestehend aus 1,3 Millionen Sätzen vermischt.

## 3.1 Herkunft der Daten

Um den Problemkontext bestmöglich darstellen zu können, wurden die Daten aus dem firmeneigenen Ticketsystem, das den Hersteller mit den Fachhändlern verbindet, extrahiert. Das genannten Ticketsystem dient dazu, Probleme von Kunden, die Fachhändler nicht selbstständig lösen können, beziehungsweise Probleme, welche auf Garantiefälle und Fehlproduktionen zurückgeführt werden können, zu diskutieren und zu analysieren. Zum Zeitpunkt der Datenerhebung beinhaltete dieses System ca. 7000 Tickets. Die eigentlichen Daten wurden aus dem Textfeld „Vorfall/Beschreibung“ entnommen, da sich dort alle in ganzen Sätzen geschriebenen Texte befanden.

## 3.2 Extraktion mittels Webcrawler

Die Extraktion der Daten vom Ticket-System fand mittels Webcrawler in Python statt. Jedes Ticket des Systems wurde systematisch durchlaufen und die nötigen Texte extrahiert. Anschließend wurden die Texte in einer Liste gespeichert, zu einem Pandas Dataframe zusammengefügt und dieser am Ende als CSV-Datei gespeichert.

### 3.2.1 Python-Code

Die programmatische Umsetzung des Webcrawlers fand mittels des Tools „Selenium“ und „Chromedriver“ statt. Diese wurden ausgewählt, um während der Erhebung visuelles Feedback zu bekommen.

Im ersten Schritt findet der Aufruf der Seite inklusive Log-In statt. Dazu wird das aus der Bibliothek stammende Modul „webdriver“ importiert. Anschließend wird der Treiber mittels „ChromeDriver“ initialisiert. Nach dem Aufruf des Ticket-Systems wird das Log-In-Formular ausgefüllt und bestätigt. Nach einer kurzen Wartezeit wird nun zur Übersichts-Seite der Tickets navigiert. Danach folgt die Hauptaufgabe des Crawlers. Mit Hilfe der ersten For-Schleife wird über alle Seiten iteriert, um zu jedem Ticket navigieren zu können. Die Tickets werden in Tabellenform auf der Seite dargestellt. Deswegen dient das Table-Body-Element, sowie die Tabellenzeilen, als Ausgangspunkt für den Zugriff auf die einzelnen Tickets. Nach dem Aufruf eines Tickets werden alle Text-Areas in der inneren Schleife durchlaufen, bis das gewünschte Eingabefeld erreicht ist und ausgelesen werden kann. Der Inhalt dieses Feldes wird ausgelesen und in einer Liste abgespeichert. Als letzten Schritt der inneren Schleife wird die Übersichts-Seite erneut aufgerufen. Die finale Zeile der äußeren Schleife bewirkt einen Wechsel auf die nächste Seite der Übersicht.

```
for i in range(183):

    tbody = driver.find_element_by_id("pageForm:ticketTable:tb")
    length = len(tbody.find_elements_by_tag_name("tr"))
    trs = tbody.find_elements_by_tag_name("tr")

    for tr in range(length):

        td = trs[tr].find_elements_by_tag_name("td")[3]
        td.find_elements_by_tag_name("a")[0].click()

        for element in driver.find_elements_by_tag_name("textarea"):
            if "customAttributesTicketCommtextValue" in element.get_attribute(
"id"):
                desc = element
                sentences_list.append(desc.get_attribute("innerHTML"))

        driver.get(url_overview)

        tbody = driver.find_element_by_id("pageForm:ticketTable:tb")
        trs = tbody.find_elements_by_tag_name("tr")

    driver.find_elements_by_class_name("chevron_right")[0].click()
```

**Code 1: Extraktion von Einträgen aus dem Ticketsystem der Firma HORSCH mittels eines Webcrawlers. Benutzt wurde das Chromedriver-Framework.**

Sobald die Texte aus allen Tickets extrahiert wurden, wird die Liste mit den gespeicherten Texten in einen „DataFrame“ der Bibliothek „Pandas“ geladen und dieser abschließend als CSV-Datei abgespeichert.

## 3.3 Vorverarbeitung

Die Vorverarbeitung der Daten für das Training verschiedener Modelle gliederte sich in vier Punkte. Zu aller erst mussten aus allen Dokumenten/extrahierten Konstrukten möglichst viele korrekte Sätze extrahiert werden. Dazu wurde im ersten Schritt versucht eine programmatische Lösung zu finden, vollständige Sätze zu extrahieren. Im nächsten Schritt folgte eine vollständige händische Durchsicht der Daten, um nun völlig sicher zu sein, dass lediglich vollwertige Sätze im Datensatz verbleiben. Anschließend mussten noch Abkürzungen jeglicher Art aus den Daten entfernt werden, um Differenzen und Übersetzungsfehler auszugleichen.

### 3.3.1 Satz-Extraktion mit Hilfe eines Dependency-Parsers

Bei der programmatischen Extraktion von Sätzen wurde auf das sogenannte „dependency parsing“ zurückgegriffen. Dabei wird der Input in seine Einzelteile zerlegt und anschließend in eine Baumstruktur überführt, welche die grammatikalischen Abhängigkeiten zwischen den einzelnen Bestandteilen eines Satzes widerspiegelt. Betrachtet man nun Ansätze, wie die Valenz-/Dependenzgrammatik kann man im Idealfall mit dieser Methode ganze Sätze aus dem Input extrahieren. Das Valenz- oder Dependenzgrammatikmodell von Lucien Tesnière stellt Verben in den Mittelpunkt eines jeden Satzes. (Osborne & Gerdes, 2019) Dabei wird ermittelt, welche Zusatzinformation in einem Satz zum Verb gehören und welche dieser Ergänzungen obligatorisch sind. (ZUM Deutsch Lernen, 2020) Im Falle der Datenerhebung dieser Arbeit wurde lediglich betrachtet, ob es sich bei der Wurzel des Inputs um ein Verb handelt oder nicht. Wurde die Wurzel als Verb identifiziert, so wurde der potentielle Satz in den Datensatz übernommen.

Zur programmatischen Umsetzung in Python wurde die Bibliothek „Spacy“ verwendet. Diese verfügt über das NLP-Model „de_core_news_lg“, welches als ein CPU-optimiertes Modell zur Verarbeitung deutscher Wörter/Sätze trainiert worden ist. Alle potentiellen Sätze, die vorher extrahiert wurden, konnten nun mit Hilfe dieses Modells analysiert werden. Dabei wurde jeder Token des Inputs durchsucht bis die Wurzel gefunden wurde. Anschließend wurde auf das POS-Tag der Wurzel zugegriffen. Damit konnte überprüft werden, ob es sich um ein Verb handelt. Die Sätze, auf die das zutraf, wurden in einer Liste gespeichert und für alle weiteren Schritte verwendet.

```
doc = nlp(desc.get_attribute("innerHTML"))
for token in doc:
    if(token.dep_ == "ROOT" and token.tag_.startswith("V")):
        sentences_list.append(desc.get_attribute("innerHTML"))
        break
```

**Code 2: Dependency-Parsing. Filtern der analysierten Einträge nach der Wurzel (ROOT), sowie Überprüfung ob es sich um ein Verb handelt.**

### 3.3.2 Händische Durchsicht

Nach der ersten Filterung der Daten mittels eines „Dependency Parsers“ folgte nun eine vollständige Durchsicht der Daten. Diese hatte das Ziel, dass nur korrekte (deutsche) Sätze im Datensatz verbleiben. Außerdem sollte die korrekte Formatierung des

Datensatzes sichergestellt werden und unerwünschte Sonderzeichen, wie zum Beispiel Anführungszeichen, entfernt werden. Ergebnis der Durchsicht ist eine Textdatei, in der sich pro Zeile ein einzelner Satz befindet, der als Input für Training, Test und Evaluation dienen kann.

### 3.3.3 Entfernen von Abkürzungen

Nachdem sichergestellt wurde, dass sich nur korrekte Sätze im Datensatz befinden, musste ebenfalls beachtet werden, dass jegliche Abkürzungen, die bei der maschinellen Übersetzung ins Englische Probleme verursachen könnten, entfernt werden. Dabei wurde ein Skript verwendet, dass alle Abkürzungen erkennen sollte. Diese wurden mit einer vordefinierten Liste verglichen und anschließend durch das ausgeschriebene Wort, dass sich in einer weiteren Liste befand, ersetzt. Wenn unbekannte Abkürzungen auftauchten, wurde ein Input der ausführenden Person verlangt, um das korrekte ausgeschriebene Wort in der Liste zu ergänzen. Somit wurden die Listen der Abkürzungen und Verbesserungen stetig erweitert.

```
abbrevs = ["Lt.", "lt.", "Stk.", "stk.", "wg.", "bzw."]
words = ["Laut", "laut", "Stück", "Stück", "wegen", "beziehungsweise"]


for i in range(len(lines)):
    for j in range(len(lines[i]) - 1):
        if "." in lines[i][j]:
            if lines[i][j] in abbrevs:
                lines[i][j] = words[abbrevs.index(lines[i][j])]
            else:
                print(" ".join(lines[i]) + " | " + lines[i][j])
                inp = input()
                if inp == "":
                    lines[i][j] = lines[i][j]
                else:
                    if lines[i][j] not in abbrevs:
                        abbrevs.append(lines[i][j])
                        words.append(inp)
                    lines[i][j] = inp
```

**Code 3: Dynamische Entfernung von Abkürzungen. Erkennt der Algorithmus eine unbekannte Abkürzung, wird der Benutzer aufgefordert eine ausgeschriebene Form einzugeben.**

### 3.3.4 Übersetzung ins Englische

Die Übersetzung der Daten musste maschinell stattfinden, da der Zeit- beziehungsweise Kostenaufwand zu groß gewesen wäre. Dazu musste ein Übersetzer gewählt werden, der möglichst viele korrekte Übersetzungen tätigt. Die Wahl fiel dabei auf den Anbieter „DeepL“. „DeepL“ ist ein Unternehmen aus Köln, das seit August 2017 einen Übersetzer anbietet, welcher auf Basis neuronaler Netze arbeitet. Das Unternehmen machte in Eigeninitiative Blindtests mit professionellen Übersetzern. Diesen Experten wurden vier, von verschiedenen Anbietern stammende, Übersetzungen vorgelegt. Sie mussten dann entscheiden, welche Übersetzung die beste ist. Nachdem diese Tests mit 119 Absätzen aus unterschiedlichen Bereichen abgeschlossen wurden, stellte man fest, dass der Übersetzer von „DeepL“ die meisten als korrekt bewerteten Übersetzungen lieferte. (DeepL, o.D)

Für die Übersetzungen des Datensatzes wurde die API von „DeepL“ verwendet. Dazu wurde jeder Satz aus der Textdatei einzeln an die API geschickt und danach in einer neuen Datei gespeichert.

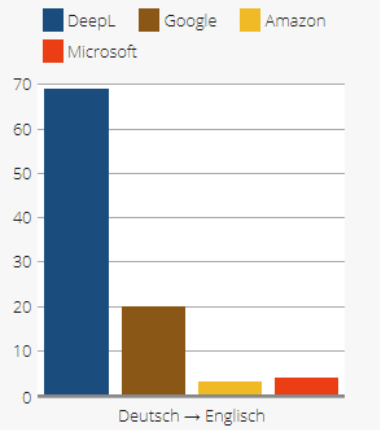


**Abbildung 18: Ergebnisse der Blindtests zu Vergleich von DeepL mit anderen Übersetzungssystemen. (https://www.deepl.com/press.html, 18.04.2021)**

```
for line in lines:
    buffer = BytesIO()
    crl = pycurl.Curl()
    crl.setopt(crl.URL, url)
    data = {
        "auth_key" : auth_key,
        "text" : line,
        "target_lang" : target_lang
    }
    pf = urlencode(data)
    crl.setopt(crl.POSTFIELDS, pf)
    crl.setopt(crl.WRITEDATA, buffer)

    crl.perform()
    crl.close()

    body = buffer.getvalue().decode("UTF-8")
    data = json.loads(body)
    translations.append(data["translations"][0]["text"])
```

**Code 4: Übersetzung der Sätze des Datensatzes mittels der API von DeepL.**

Der vollständige HORSCH Trainings- und Testdatensatz besteht aus 5770 Sätzen.

Nach diesem Schritt fand die Vermischung des HORSCH Datensatzes mit den 1,3 Millionen weiteren Sätzen statt um den Datensatz zu vergrößern. Die Daten wurden von Tatoeba[1] bezogen.

## 3.4 Fehlerinjektion

Die Fehlerinjektion für Training, Test und Evaluation fand, wie die Übersetzung, maschinell statt, da für diese Arbeit keine natürlichen fehlerhafte Konstrukte zur Verfügung standen. Die ursprünglich extrahierten Daten waren, falls es sich um fehlerhafte Auszüge handelte, stichpunktartig verfasst und bildeten somit keine Grundlage für natürliche grammatikalische Fehler.

Für das maschinelle injizieren von Fehlern wurde das Skript „errorify“ aus dem GitHub-Repository „PIE“ von Abhijeet Awasthi verwendet. Das „PIE“ Modell für grammatical error correction wird auch im weiteren Verlauf dieser Arbeit verwendet, nachtrainiert und evaluiert. Das Skript injiziert Fehler von Verbformen, Nomen, Worteinsätze, Löschungen und Verwechslungen. Die Verbformen und Veränderungen von Nomen

[1] https://tatoeba.org/deu/downloads

werden aus der Datei „morphs.txt“ bezogen. Die Basis dieser Datei stammt aus einem separaten GitHub-Repository von „ixa-ehu“[2]. Die INSERTS, DELETES und REPLACES stammen aus den öffentlich verfügbaren Datensätzen Lang-8, NUCLE und FCE.
Der Output dieses Skripts lieferte zwei parallele Dateien, eines mit den korrekten und das andere mit den fehlerhaften Sätzen. Gegenübergestellt sehen diese wie folgt aus:

| **Fehlerhafte Sätze** | **Korrekte Sätze** |
|---|---|
| Due to the tolerance-related height deviations of Then up for 1 cm, there are always problem with the packer wipers in the Express KR not working cleanly. | Due to the tolerance-related height deviation of up to 1 cm, there are always problems with the packer wipers in the Express KR not working cleanly. |
| When folding, the screw collided and the tank damaged accordingly. | When folding, the screw collides and the tank is damaged accordingly. |
| A lot of air were lost at the calibration flap. | A lot of air is lost at the calibration flap. |
| He may be able to cut out a hand larger hole and an adapter plate with between. | He may be able to cut out a larger hole and make an adapter plate in between. |
| Where do we go from here? | Where do we go from here? |

**Tabelle 1: Gegenüberstellung von fehlerhaften und korrekten Sätzen nach der Fehlerinjektion.**

## 3.5 Evaluationsdatensatz

Der Testdatensatz wurde unabhängig vom bisherigen Datensatz extrahiert. Die Sätze stammen aus dem Internetauftritt der Firma HORSCH und hat eine Größe von 659 Sätzen, was ca. 11,4% vom bisherigen HORSCH Datensatz ausmacht. Er wurde händisch extrahiert und formatiert. Die Satzstruktur dieser Sätze ist meist etwas komplexer und es finden sich alle landwirtschaftlichen Fachbegriffe und firmeneigene Bezeichnungen in diesen Sätzen.

[2] https://github.com/ixa-ehu/matxin/blob/master/data/freeling/en/dictionary/verbs

# 4 Verwendete Modelle

Das folgende Kapitel beschäftigt sich mit den für das Nachtrainieren verwendeten Modellen. Für jedes Modell wird der Aufbau, Vorgehen und Besonderheiten beim Preprocessing, sowie die verwendeten Parameter für das Nachtrainieren eingegangen. Im Anschluss werden die Modelle verglichen und auf deren Effektivität eingegangen.
Für die Auswahl der Modelle wurden mehrere Kriterien betrachtet. Das wohl wichtigste Merkmal war die Performance auf dem CoNLL-2014 shared task Testdatensatz. (Ruder, o.D.) Die Performance wurde anhand des $M^2$-Scores gemessen, der in dieser Arbeit ebenfalls Anwendung findet und erklärt wird. Als weitere Eigenschaft sollte die Architektur der Modelle herangezogen werden. Diese sollten sich in wesentlichen Punkten unterscheiden.

## 4.1 Grammatical Error Correction: Tag, Not Rewrite

Als eines der führenden Architekturen im Gebiet grammatical error correction, dessen Code und Modelle open-source verfügbar sind, ist GECToR, aus der Arbeit „Grammatical Error Correction: Tag, Not Rewrite“, bestens für diese Arbeit geeignet. Das Modell, beziehungsweise die Modelle als Ensemble, erreichen auf dem CoNLL shared task Datensatz einen $M^2$-Score von F0.5 = 65,3, bzw. F0.5 = 66,5. Ziel dieser Architektur ist es nicht Text-Sequenzen zu generieren, wie es bei vielen anderen NMT-Modellen der Fall ist. Der Ansatz dieser Arbeit war es, die Tokens der Input-Sequenzen zu annotieren, um mit diesen Tags Korrekturen durchzuführen. (Omelianchuk et al., 2020)

### 4.1.1 Aufbau des Modells

Die Architektur des Modells gliedert sich in drei Bestandteile. Als Encoder dient ein BERT-ähnlicher Transformer. BERT oder ähnliche Transformer-Modelle nutzen einen Attention-Mechanismus, der die kontextuellen Beziehungen zwischen den Wörtern einer Sequenz lernt. Diese Bidirektionalität erlaubt das Lernen des Kontexts eines Wortes auf Basis aller umgebenen Tokens. (Horev, 2018) In den Experimenten für diese Arbeit wurden lediglich drei Transformer-Modelle verwendet. Diese wurden ausgewählt, da sie in den Versuchen von GECToR die besten Ergebnisse erzielten. Der grundlegende Aufbau dieser Transformer wird in den folgenden Abschnitten beschrieben.

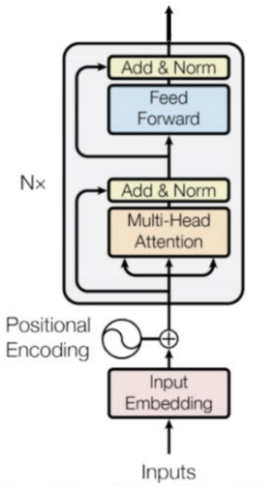


**Abbildung 19: Aufbau eines Transformer-Encoders am Beispiel BERT (Vaswani et al., 2017)**

BERT ist folgendermaßen aufgebaut:

Nach dem Token Embedding folgt der Encoder-Stack mit sechs identischen Schichten. Jeder dieser Layer hat zwei Sub-Layer. Der erste Layer ist ein Multihead-Attention-Layer. Auf diesen folgt fully conntected Feed-Forward-Netzwerk. Um diese zwei Sub-layer wurde eine Residual Connection gelegt (Vaswani et al., 2017), die es Gradienten erlaubt, direkt durch ein Netzwerk zu fließen, ohne nicht-lineare Aktivierungsfunktionen passieren zu müssen. (He et al., 2016) Abschluss des BERT-Encoders ist eine Layer Normalization. Diese wird unter anderem dazu verwendet, die Trainingszeit zu reduzieren und die Dynamiken der hidden states zu stabilisieren. (Ba et al., 2016)

RoBERTa ist ein BERT-basiertes Modell. RoBERTa lässt im Gegensatz zu BERT die Next Sentence Prediction außen vor und fügt stattdessen dynamic masking ein. Damit wurde erreicht, dass sich der verschleierte Token während des Trainings in den Epochen verändern kann. RoBERTa wurde ausserdem mit einem größeren Datensatz trainiert. (Liu et al., 2019)

Der dritte verwendete Transformer ist XLNet. XLNet verwendet permutation language modeling. Das bewirkt, dass alle Tokens in zufälliger Reihenfolge vorhergesagt werden, was wiederum dazu führt, dass das Model bidirektionale Beziehungen und Abhängigkeiten gut erlernen kann. Als Basisarchitektur von XLNet diente der Transformer XL. Die Architektur baut auf der Transformer-Architektur, die oben beschrieben wurde, auf. (Yang et al., 2019)

Nach dem Transformer Model folgten in der Architektur des Modells von GECToR zwei Linear Layer. Sie dienen der Error-Detection und dem Error-Tagging auf Token-Ebene. Die abschließende Schicht des Modells bildeten Dense-/Softmax-Layers. (Omelianchuk et al., 2020)

### 4.1.2 Vorverarbeitung des Datensatzes

Die Vorverarbeitung der Daten besteht daraus, jeden Token des Datensatzes zu annotieren. Dabei wurden die drei folgenden Hauptfehlergruppen abgedeckt.

**Transformationen auf Token-Ebene:**

Fehler auf Token-Ebene beinhalten neben Rechtschreibfehlern, Singular-/Plural-Fehler oder Fehler bei Groß- und Kleinschreibung bei Nomen, Fehler der Verbform und Fehler beim Konjugieren des Verbs, passend zum Subjekt. Für jede Art von Fehler wurde ein eigener Tag definiert. Die Tags, die Fehler der Verbformen beinhalten, wurden außerdem um Informationen über die aktuelle, sowie die gewünschte Verbform des Tokens ergänzt.

**Einfache Transformationen:**

Die einfachen Transformationen beinhalten vier Operationen: KEEP, DELETE, INSERT, REPLACE. Der KEEP-Tag wird dazu verwendet, korrekte Wörter in einem Satz zu annotieren und diese in ihrer Ausgangsform beizubehalten. DELETE findet bei überflüssigen Wörtern Verwendung und INSERT ergänzt vergessene, jedoch relevante, Wörter. Der REPLACE-Tag tauscht Wörter aus, die im Kontext des Satzes falsch sind.

**Sonstige Transformationen:**

Zudem gibt es noch Tags, die dafür zuständig sind, dass ein Token mit einem darauffolgenden Token in einem einzelnen Wort zusammengefasst wird. Es wurde ebenfalls ein Tag für die Transformation in die entgegengesetzte Richtung definiert. Dieser bewirkt, dass ein Token in zwei Wörter aufgeteilt wird.

Das Preprocessing an sich beinhaltet zwei Hauptschritte. Im ersten Schritt wurden die Source-Tokens zu einer Subsequenz an Tokens der Target-Tokens zugeordnet. Dabei wurde der kleinste Abstand der Sequenzen verwendet, der mit Hilfe einer modifizierten Levenshtein-Distanz berechnet wurde. Anschließend wurde jedem Mapping von Tokens Transformationen zugewiesen, die die Source-Sequenz in die entsprechende Target-Sequenz umwandeln. Hat die Liste der Transformationen mehr als eine

Veränderung beinhaltet, so wurde lediglich die erste Transformation der Liste übernommen. (Omelianchuk et al., 2020)

| **Source** | **Target** | **Preprocessed** |
|---|---|---|
| Don't made a promise which you cannot keep. | Don't make a promise which you cannot keep. | $STARTSEPL\|\|\|SEPR$KEEP Don'tSEPL\|\|\|SEPR$KEEP madeSEPL\|\|\|SEPR$TRANSFORM_VERB_VBN_VB aSEPL\|\|\|SEPR$KEEP promiseSEPL\|\|\|SEPR$KEEP whichSEPL\|\|\|SEPR$KEEP youSEPL\|\|\|SEPR$KEEP cannotSEPL\|\|\|SEPR$KEEP keep.SEPL\|\|\|SEPR$KEEP |

**Tabelle 2: Veranschaulichung des Ergebnisses der Vorverarbeitung für die GECToR-Modelle.**

### 4.1.3 Verwendete Parameter

Für das erste Nachtrainieren der Modelle von GECToR wurden hauptsächlich die Standardparameter verwendet. Die Pfade für Train-, Dev-Set und Model-Directory wurden entsprechend angepasst. Alle online verfügbaren und vortrainierten Modelle wurden heruntergeladen. Um die einzelnen Modelle nun nachtrainieren zu können musste der Parameter „transformer_model“ für jedes Modell angepasst werden. Als Transformer-Modelle fanden die oben genannten (BERT, RoBERTa und XLNet) Verwendung. Die Anzahl der Epochen wurde für erste Tests auf 4 gesetzt. Nach diesen vier Epochen fand die erste Evaluation für alle Modelle statt, um zu sehen, ob und wie sie sich beim Nachtrainieren verbessern. Die Batch-Size musste aufgrund fehlender Ressourcen auf meinem System auf 32 herabgesetzt werden. Die maximale Länge eines einzelnen Satzes blieb auf dem Standard-Wert 50. Als Vokabular des Modells wurde das ursprüngliche Vokabular benutzt. Der Pfad und Größe des Vokabulars blieben deshalb unverändert. Alle weiteren Parameter wurden mit dem gegebenen Standard-Wert initialisiert.

### 4.1.4 Ensemblebildung

GECToR bietet die Möglichkeit aus den trainierten Modellen Ensembles zu bilden, mit denen anschließend Vorhersagen durchgeführt werden können. Die Vorhersagen der Ensembles entstehen aus dem Bilden des Durchschnitts der Vorhersagewahrscheinlichkeit der einzelnen Komponenten. (Omelianchuk et al., 2020)
Diese Möglichkeit möchte ich in dieser Arbeit ebenfalls nutzen. Nach der ersten Evaluation und der Auswahl der erfolgreichsten Modelle und weiteren 16 Epochen Training werde ich die Modelle miteinander kombinieren und die Ensembles ebenfalls evaluieren.

## 4.2 Parallel Iterative Edit Models for Local Sequence Transduction

Das Modell PIE aus der Arbeit Parallel Iterative Edit Models for Local Sequence Transduction erreicht ebenfalls State-of-the-Art-Ergebnisse. Auf dem CoNLL-2014 shared task Datensatz erreicht dieses Modell einen F0.5-Score von 61,2. Genau wie GECToR verfolgt PIE den Ansatz, Sequenzen zu labeln, anstatt Sequenzen zu generieren. Ziel dieser Arbeit war es auch, die Model-Inference zu beschleunigen. Das wurde durch paralleles Decoding der Sequenzen erreicht. PIE erreicht bis zu 15-Mal schnellere Vorhersagen als andere Encoder-Decoder-basierte Modelle. (Awasthi et al., 2019)

### 4.2.1 Aufbau des Modells

Um Änderungen in einem Satz zur Korrektur vorherzusagen, wurde für dieses Modell ein BERT-ähnlicher Encoder adaptiert. Das Transformer Modell wurde angepasst, indem die Logit-Schicht über die Edit-Befehle und deren Token faktorisiert wird. Abgeschlossen wird das Modell von einem Softmax-Layer, der dazu dient Wahrscheinlichkeiten über den Raum der Edits vorherzusagen. Durch iterative Verfeinerung der Ergebnisse wird sichergestellt, dass die Inferenzkapazität erhöht wird. (Awasthi et al., 2019)

### 4.2.2 Vorverarbeitung der Daten

Das Ergebnis der Vorverarbeitung sind vier Hauptdateien. Die korrekten und inkorrekten Sätze werden einzelne Tokens aufgeteilt. Außerdem wird jeder Satz mit einem Label für den Sequenzstart und für das Ende der Sequenz versehen. Die Tokens werden mit Leerzeichen zusammengefügt und anschließend in den Dateien „train_corr_tokens.txt“

sowie „train_incorr_tokens.txt“ abgespeichert. Des Weiteren werden die Tokens aller Sätze in numerische IDs mittels des Vokabulars des Modells aufgelöst. Die numerischen Repräsentationen der Sätze werden in der Datei „train_incorr.txt“ gespeichert. Da es auch in diesem Modell um die Vorhersage von Labels/Edits an den Tokens und Sätzen geht, müssen dem Modell auch dieses Mal die Annotationen der Satzpaare zum Training bekannt sein. Diese sind in der Datei „train_labels.txt“ abgelegt. Die Labels und die dazugehörigen Operationen sind mit Hilfe von numerischen IDs umgesetzt worden. Zur Berechnung der Labels/Operationen wurde auch hier der Levenshtein Distanz Algorithmus verwendet, um die Token Paare identifizieren zu können. Anschließend wurden die Transformationen entsprechend dem edit space, welcher aus copy, delete, 1000 inserts, 1000 replaces und 29 verschiedenen Transformationen auf Token-Ebene besteht, annotiert. Inserts und Replaces bestehen hauptsächlich aus Artikeln, Pronomen, Präpositionen, Konjunktionen und Verben. Die Transformationen der Tokens geschehen größtenteils auf Suffix-Ebene. Die Transformationen des edit spaces stammen aus den häufigsten Transformationen aus den Trainingsdaten. Als ursprüngliche Trainingsdaten wurden die drei öffentlich verfügbaren Datensätze Lang-8, NUCLE und FCE verwendet. (Awasthi et al., 2019)

| **Source** | **Target** | **Source_tokens** | **Target_tokens** | **Source_IDs** | **Labels** |
|---|---|---|---|---|---|
| Don't made a promise which you cannot keep. | Don't make a promise which you cannot keep. | [CLS] Don ' t made a promise which you cannot keep . [SEP] | [CLS] And where is the money coming from ? [SEP] | 101 1262 1187 1116 1138 1948 1682 1909 1121 136 102 | 3 3 3 1008 1007 3 4 3 3 3 3 |

**Tabelle 3: Veranschauchlichung des Ergebnisses der Vorverarbeitung für das Modell PIE.**

### 4.2.3 Verwendete Parameter

Hier wurden, ähnlich wie bei GECToR, größtenteils Standardparameter verwendet. Alle Pfade zu Vokabular, Model, Trainings- und Testdatensatz wurden entsprechend angepasst. Obwohl die Urheber mehrere Architekturen und Größen ihres Modells im Paper

beschreiben, steht nur eines davon zum Download zur Verfügung. Das verwendete Modell wurde mit folgenden Hyperparametern initialisiert:

- Attention_probs_dropout_prob = 0,1
- Directionality = bi-directional
- Hidden_act = gelu
- Hidden_dropout_prob = 0,1
- Hidden_size = 1024
- Initializer_range = 0,02
- Intermediate_size = 4096
- Max_position_embeddings = 512
- Num_attention_heads = 16
- Num_hidden_layers = 24
- Type_vocab_size = 2
- Vocab_size = 28996
- Copy_weight = 0,4

Nachzulesen sind die oben genannten Hyperparameter im Appendix des offiziellen Papers zum Code/Modell.
Verändert wurden lediglich zwei Parameter. Zum einen wurde die batch_size aufgrund von Limitierungen durch die verwendeten Ressourcen für diese Arbeit auf einen Wert von 16 reduziert. Zum anderen wurde die Anzahl der Epochen auf 4 gesetzt, um dieselbe Ausgangssituation für die erste Evaluation der Modelle zu erhalten.

## 4.3 Fairseq-GEC

Das Modell von Fairseq-GEC verwendet als Basis einen menschlichen Ansatz der Grammatikkorrektur. Die Grundidee ist es, korrekte/unveränderte Wörter von der Quelle zum Zielsatz zu kopieren. Mit diesem Ansatz lassen sich zwei Probleme zufriedenstellend lösen. Zum einen wird das Problem von Out-Of-Vocabulary-Words umgangen, indem man sie einfach in den Zielsatz kopiert. Zum anderen wird durch das Kopieren der korrekten Wörter die Ladung des Modells verringert. Dadurch hat das Modell größere Kapazitäten für Änderungen der inkorrekten Wörter. Durch Vortrainieren mit Denoising-Auto-Encodern wurde in dieser Arbeit außerdem das Problem eines unzureichend gelabelten Datensatzes behandelt. Durch diese Methode konnte auf dem CoNLL 2014

Testdatensatz ein Max-Match-Score von F0.5 = 61,15 erreicht werden. Damit erreicht die Performance dieses Modells State-of-the-Art-Niveau. (Zhao, et al., 2019)

### 4.3.1 Aufbau des Modells

Die Architektur von Fairseq-GEC basiert ebenfalls auf einem Attention-basierten Transformer. Der Transformer encodiert die Sätze mit einem Stack von N identischen Blöcken. Jeder dieser Blöcke wendet eine Multihead-Self-Attention, gefolgt von einem positionellen Feed-Forward-Layer, auf den Eingangssatz an. Dieser Aufbau gewährleistet die Kontextsensitivität des Encoders. Der Decoder ist nach demselben Prinzip aufgebaut. Allerdings besitzt dieser mit einem Attention-Layer über den hidden-state des Encoders eine zusätzliche Schicht. Durch eine Softmax-Funktion zwischen dem inneren Produkt des hidden-states des Ziels und der Embedding-Matrix werden Wahrscheinlichkeiten für das nächste Wort errechnet. Der Transformer wurde aus dem FAIR Sequence-to-Sequence Toolkit übernommen. Der Vorhersage des nächsten Wortes steht der Kopiermechanismus gegenüber, der es erlaubt, Wörter aus dem Ausgangssatz zu kopieren. Die Balance zwischen Kopieren und Wortgenerierung wird durch einen Ausgleichsfaktor reguliert. (Zhao, et al., 2019)

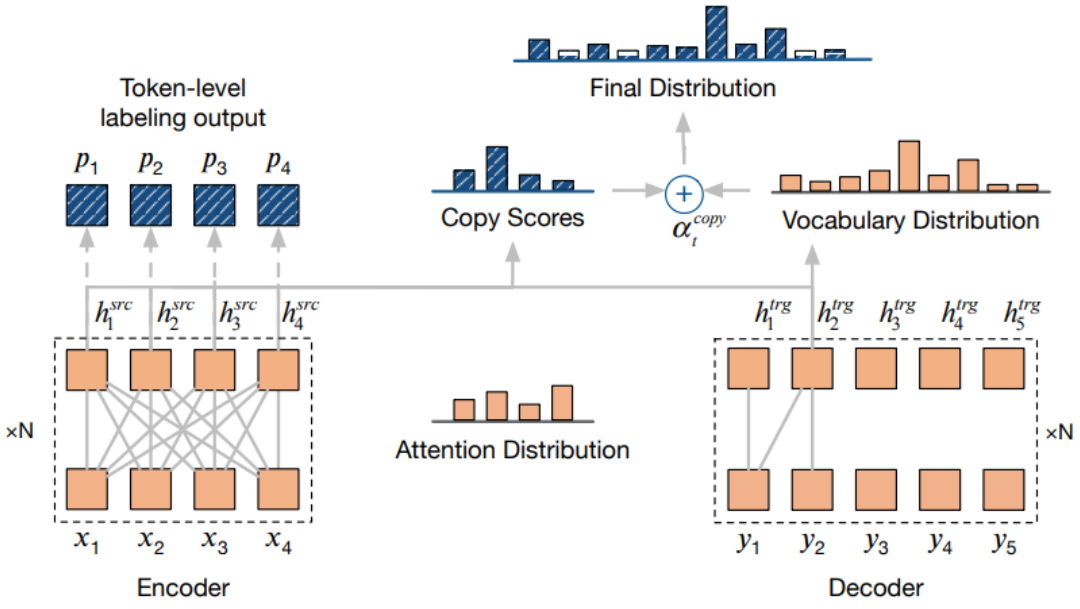


**Abbildung 20: Aufbau des Fairseq-GEC-Modells zur Veranschaulichung des Kopiermechanismus von korrekten Tokens (Zhao et al., 2019)**

### 4.3.2 Vorverarbeitung der Daten

Ergebnis der Vorverarbeitung sind zwei Ordner. Ein Ordner beinhaltet Daten als Plain-Text, der andere Ordner beinhaltet die Trainingsdaten in verschiedenen Formen. Im Ordner „data_raw“ finden sich insgesamt vier Dateien. Es sind zwei Dictionaries enthalten, die jeweils ein Mapping zwischen Source-/Target-Tokens zu einem Index enthalten. Die zwei anderen zwei Dateien beinhalten den Testdatensatz. Im Ordner „data_bin“ befinden sich 12 Dateien. Acht dieser Dateien dienen dazu, den Trainings- und Validierungsdatensatz in Binärform darzustellen. „train.label.src.txt“ und „train.label.tgt.txt“ sind die Annotationen, welche für den Kopiermechanismus benötigt werden. Jeder Token eines Satzes wurde mit 0 oder 1 gelabelt. „0“ steht für ein korrektes Wort, das kopiert werden kann. „1“ steht dabei für Tokens, die ersetzt/generiert werden müssen. Bei den letzten beiden Dateien dieses Ordners handelt es sich um dieselben Dictionaries, die auch im „data_raw“-Ordner zu finden sind. (Zhao, et al., 2019)

| Source | Target | Label (Source) | Label (Target) |
|---|---|---|---|
| Don't made a promise which you cannot keep. | Don't make a promise which you cannot keep. | 0 1 0 0 0 0 0 0 | 0 1 0 0 0 0 0 0 |

**Tabelle 4: Veranschauchlichung des Ergebnisses der Vorverarbeitung für das Modell Fairseq-GEC.**

### 4.3.3 Verwendete Parameter

Für das Nachtrainieren dieses Modells wurden lediglich zwei Parameter verändert. Die maximale Anzahl an Epochen wurde auf 4 herabgesetzt, um dieselbe Ausgangssituation, wie bei den beiden anderen Modellen für die erste Evaluation zu erreichen. Die batch_size wurde aufgrund der verwendeten Hardware auf 32 reduziert. Alle anderen Parameter wurden mit den Standardwerten initialisiert. Diese sahen wie folgt aus:

- Embedding-Dimensionen: 512
- Anzahl der Encoder-/Decoder-Layer: 6
- Attention-Heads: 8
- Innere Schicht des Feed-Forward-Netzwerks: 4096
- Droptout: 0,2
- Größe des Vokabulars: 50000
- Learning-Rate: 0,0002

Das Fairseq-GEC-Modell umfasst insgesamt 97 Millionen trainierbare Parameter. (Zhao, et al., 2019)

# 5 Erste Evaluation der Ergebnisse und Vergleich mit den ursprünglichen Modellen

In diesem Kapitel wird die erste Evaluation der Modelle nach vier Epochen mit dem Datensatz von HORSCH vorgestellt. Dazu wird der in Kapitel 3 vorgestellte Testdatensatz verwendet. Die nachtrainierten Modelle sollen mittels verschiedener Metriken mit den ursprünglichen Modellen verglichen werden, um feststellen zu können, ob sich die Modelle nach den ersten vier Epochen des Retrainings verbessert haben. Dazu sollen die Vorhersagen an den folgenden Metriken evaluiert werden.

- MaxMatch-Score ($M^2$-Score)[3]
- Google Bilingual Evaluation Understudy Score (GLEU-Score)[4]
- Sacre Bilingual Evaluation Understudy Score (SacreBLEU-Score)[5]
- Recall-Oriented Understudy for Gisting Evaluation (ROUGE-Score)[6]

Die genannten Metriken sollen im Folgenden näher erläutert werden. Anschließend werden die Ergebnisse der Evaluation nach dem ersten Nachtrainieren vorgestellt und interpretiert.

## 5.1 Verwendete Metriken

### 5.1.1 MaxMatch-Score

Der MaxMatch-Score ($M^2$-Score) ist eine Methode, die speziell für die Evaluation von GEC-Systemen entwickelt wurde. Als Eingaben für diese Metrik dienen drei Bestandteile: (1) Ein Ausgangssatz, (2) die Hypothese eines GEC-Systems und (3) ein Satz an Veränderungen in Form von gold-standard-annotations. Jeder dieser Edits kann durch ein Triplet repräsentiert werden. Zwei Bestandteile dieses Triplets sind Start- und Endtoken des jeweiligen Edits. Der dritte Bestandteil ist ein Satz aus ein oder mehreren möglichen Korrekturen. (Dahlmeier & Ng, 2012)

Als ersten Schritt der Evaluation mit dem $M^2$-Score muss man seinen Testdatensatz mit den genannten Annotationen versehen. Um dies zu tun wurden die Python-Bibliothek

[3] https://github.com/nusnlp/m2scorer
[4] https://github.com/keisks/jfleg
[5] https://pypi.org/project/sacrebleu/
[6] https://pypi.org/project/rouge/

*Errant* benötigt. Zunächst wurden die Ausgangs- und Zielsätze des Testdatensatzes in Listen gespeichert. Als nächstes benötigte man ein Annotator-Objekt, welches für das Generieren der Edits verantwortlich ist. Anschließend wurde über die Listen an korrekten/inkorrekten Sätzen iteriert, um die Annotation generieren zu lassen und in der Datei „source_gold.txt“ abgespeichert. Das Ergebnis sieht wie folgt aus:

| **Source** | **Target** | **Annotationen** |
|---|---|---|
| The user always retains absolute control data. | The user always retains absolute control over his data. | S The user always retains absolute control data.<br>A 6 6\|\|\|M:OTHER\|\|\|over his\|\|\|REQUIRED\|\|\|-NONE-\|\|\|0 |
| This more transmits via the sophisticated machine hydraulics. | This is transmitted via the sophisticated machine hydraulics. | S This more transmits via the sophisticated machine hydraulics.<br>A 1 2\|\|\|R:OTHER\|\|\|is\|\|\|REQUIRED\|\|\|-NONE-\|\|\|0<br>A 2 3\|\|\|R:VERB:FORM\|\|\|transmitted\|\|\|REQUIRED\|\|\|-NONE-\|\|\|0 |

**Tabelle 5: Veranschauchlichung der Source-Gold-Annotation, die für den MaxMatch-Score verwendet wird.**

Für die Erstellung dieser Datei wurde folgender Code verwendet:

```
annotated_sentences = ""
for i in range(len(corr_sentences)):
    orig = annotator.parse(incorr_sentences[i])
    cor = annotator.parse(corr_sentences[i])
    edits = annotator.annotate(orig, cor)

    source_gold = "S {}\n".format(incorr_sentences[i])
    for e in edits:
        source_gold += e.to_m2() + "\n"
    source_gold += "\n"
    annotated_sentences += source_gold
```

**Code 5: Überführen von Ausgangs- und Zielsätzen in die Source-Gold-Annotation.**

Zur eigentlichen Evaluation wird nun ein Satz von Veränderungen zwischen den Ausgans- und Hypothesensätzen des Systems gebildet. Anhand dieses Sets lässt sich Precision, Recall und F-Score zwischen den Edits und den Annotationen errechnen.

### 5.1.2 GLEU-Score

Der GLEU-Score ist eine Metrik, die von Google ins Leben gerufen wurde, um die Schwächen des BLEU-Score bei der Evaluation von einzelnen Sätzen zu kompensieren. Für die Berechnung des GLEU-Scores werden n-Grams der Ordnung n = {1, 2, 3, 4} gebildet. Anschließend wird der Recall durch Division der Anzahl der gemeinsamen n-Grams durch die vollständige Anzahl an n-Grams errechnet. Die Kalkulation der Precision-Metrik erfolgt durch Teilen der gemeinsamen n-Grams durch die Anzahl aller n-Grams der Vorhersage. Der GLEU-Score selbst ist das Minimum von Recall und Precision. Damit ist das Ergebnis immer zwischen 0 (keine Übereinstimmungen) und 1 (vollständig übereinstimmend). Die Arbeit von Google zeigt auch, dass der GLEU-Score stark mit dem BLEU-Score korreliert, sich jedoch in den Nachteilen auf Satz-Ebene hervorhebt. (Wu et al., 2016)

Zur Berechnung des GLEU-Score in dieser Arbeit wurde auf die Arbeit von Courtney Napoles, Keisuke Sakaguchi und Joel Tetreault zurückgegriffen. Im GitHub-Repository zu dieser Arbeit[7] befindet sich ein Python-Script zur Berechnung des GLEU-Scores[8].

### 5.1.3 Sacre Bilingual Evaluation Understudy Score

Der SacreBLEU-Score ist eine Variante des BLEU-Scores, die diesen vergleichbarer machen soll. Der BLEU-Score ist eine Metrik, die häufig im Feld Neural-Maschine-Translation (NMT) verwendet wird. (Papineni et al., 2002) Dieser stellt weniger eine einzelne Metrik, sondern eine Reihe parametrisierter Methoden dar. Diese Parameter beinhalten:

- Die Anzahl der benutzten Referenzen
- Multi-Referenz-Parameter zur Berechnung des Length-Penalty, der dazu verwendet wird, zu kurze oder zu lange Übersetzungen zu bestrafen.
- Die maximale n-Gram-Länge
- Glättungsparameter

Der SacreBLEU-Score vereinheitlicht diese Parameter, um Vergleichbarkeit zwischen verschiedenen Arbeiten zu gewährleisten und so eine Basis zu schaffen, mit der man Systeme vergleichen kann. (Post, 2018)

[7] https://github.com/christophmeyer13/Bachelorarbeit
[8] https://github.com/keisks/jfleg

Grammatical Error Correction steht in starker Relation zur NMT, da man GEC als Übersetzung einer Sequenz von einer (inkorrekten) in eine (korrekten) Sprache sehen kann. (Ailani et al., 2019) Deswegen kann der BLEU-Score zur Evaluation von GEC-Systemen genutzt werden.

Die Berechnung des BLEU-Scores kann auf eine modifizierte Variante der Precision-Metrik zurückgeführt werden. Als erstes werden die gemeinesamen Vorkommen an n-Grams, die sowohl in der Referenz als auch in der Hypothese vorkommen, durch die Anzahl der n-Grams in in der Hypothese dividiert. Diese Division wird von n=1 bis n=4 durchgeführt und die Ergebnisse multipliziert. Zum Abschluss wird das Resultat mit dem Berivity-Penalty (Length-Penalty) multipliziert. Der BP ist das Minimum von 1 und der Anzahl der Tokens der Hypothese dividiert durch die Anzahl der Tokens des Referenzübersetzung. (Papineni et al., 2002)

$$BLEU - 4 = \min\left(1, \frac{H}{R}\right) \prod \frac{\#(n - Grams\ aus\ h\ und\ r)}{\#(n - Grams\ aus\ h)}$$

### 5.1.4 Recall-Oriented Understudy for Gisting Evaluation

Der ROUGE-Score ist eine Metrik, die ursprünglich zur Bestimmung der Qualität von Textzusammenfassungen entwickelt wurde. Für diese Arbeit ist lediglich die Metrik ROUGE-L relevant, da sie auf Satzebene arbeitet und gezeigt wurde, dass ROUGE-L für die Evaluation von Machine Translation Tasks geeignet ist. (Lin, 2004) Bei der Evaluation mittels ROUGE-L geht es um die *longest common subsequence* (LCS) in einem Satz. Hierbei wird gezählt, wie viele Tokens eines Satzes der Hypothese mit den Bestandteilen eines Referenzsatzes übereinstimmen. Anhand dieser LCS lässt sich nun Precision, Recall und F-Score berechnen, um ein System zu evaluieren.

Bei der Berechnung des Precision-Scores wird die ermittelte LCS durch die Anzahl an Tokens der Vorhersage geteilt, wobei H die Hypothese und R der Referenzsatz ist.

$$P_{LCS} = \frac{LCS(H, R)}{len(H)}$$

Der Recall wird durch Division der LCS mit der Länge des Referenzsatzes errechnet.

$$R_{LCS} = \frac{LCS(H, R)}{len(R)}$$

Der F-Score kann anschließend aus Recall und Precision berechnet werden. Dabei gibt β an, welchen Beitrag der Recall an der Berechnung des F-Score hat. Wird β verringert, hat der Recall einen geringen Einfluss am F-Score und umgekehrt. (Lin & Och, 2004)

$$\frac{(1 + \beta^2) * R_{LCS} * P_{LCS}}{R_{LCS} + \beta^2 * P_{LCS}}$$

## 5.2 Einordnung der Ergebnisse

Der folgende Abschnitt ist in zwei Teile aufgeteilt. Zuerst soll anhand der oben beschriebenen Metriken evaluiert werden, ob sich die Modelle durch das Nachtrainieren mit dem Datensatz steigern konnten. Dazu werden im ersten Abschnitt alle Modelle mit den ursprünglichen Modelllen einzeln verglichen. Im zweiten Teil der Evaluation soll anschließend evaluiert werden, welche Modelle die besten Ergebnisse liefern, um diese Modelle für das weitere Training auszuwählen. Dadurch sollte die enorme Gesamttrainingszeit reduziert werden.

### 5.2.1 Vergleich zwischen den vortrainierten mit den entsprechenden nachtrainierten Modellen

Zuerst soll nun gezeigt werden, wie sich das Training der Modelle von GECToR auf die Perfomance dieser Modelle ausgewirkt hat. Das Ergebnis des Trainings waren drei Modelle, welche die oben beschriebenen Transformer-Modelle implementieren. Jedes der Modelle soll mit seinem entsprechenden Ausgangsmodell verglichen werden. Die Ergebnisse werden sowohl in tabellarischer Form als auch in Form von Diagrammen dargestellt, beschrieben und anschließend interpretiert.

**Transformer-Modell BERT:**

Ausgansbasis des BERT-Modells war die Performance, gemessen am M²-Score, von F0.5 = 63,0 auf dem CoNNL-2014 Testdatensatz. Auf dem Testdatensatz dieser Arbeit erreichte dieses Modell einen F0.5-Score von 55,31. Die Verschlechterung der Performance auf diesem Datensatz könnte auf die Komplexität und das fachspezifische Vokabular zurückzuführen sein. Nach vier Epochen Retraining konnte das Modell seine Performance um +8,93 steigern. Das resultiert in einem F0.5 von 64,24.

| **M²-Score** |
|---|

| **Transformer-Modell** | **Precision** | **Recall** | **F0.5** |
|---|---|---|---|
| BERT (original) | 60,79 | 40,65 | 55,31 |
| BERT (retrained) | 67,14 | 54,76 | 64,24 |

**Tabelle 6: Vergleich des nachtrainierten GECToR-BERT-Modells mit dem ursprünglichen Modell mit Hilfe des $M^2$-Scores.**

Das Nachtrainieren dieses Modells wirkte sich sowohl positiv auf die Vorhersagepräzision mit einer Steigerung von 6,35 Prozent, als auch auf die Anzahl der korrekt identifizierten Fehler mit einem Anstieg von 14,11 Prozent aus.

Das Ergebnis der Evaluation durch die GLEU-Metrik verlief ebenfalls sehr positiv. Das Training erbrachte eine Leistungssteigerung von 6,01 Prozent. Das Ausgangsmodell erreichte einen GLEU-Score von 71,32. Das nachtrainierte System kam auf dem HORSCH Testdatensatz auf einen Wert von 77,33.

| **GLEU-Score** | |
|---|---|
| **Transformer-Modell** | **GLEU** |
| BERT (original) | 71,32 |
| BERT (retrained) | 77,33 |

**Tabelle 7: Vergleich des nachtrainierten GECToR-BERT-Modells mit dem ursprünglichen Modell mit Hilfe des GLEU-Scores.**

Bei der Bewertung durch den SacreBLEU-Score sind keine signifikanten Änderungen zu beobachten. Das könnte am hohen Ausgangsscore liegen. Durch den Vergleich der Vorhersage mit dem Referenzsatz, bei dem pro Satz nur geringe Veränderungen vorgenommen werden mussten, was auf die geringe Fehleranzahl zurückzuführen ist, kommt die Abschätzung zur n-Gram Metriken auf verhältnismäßig hohe Ergebnisse. Bei dieser Metrik konnte durch Training des Modells nur eine Steigerung von 2,62 Prozentpunkten erreicht werden. Der erreichte Wert des Ausgangsmodells lag bei 82,01. Das nachtrainierte System konnte einen Wert von 84,63 erreichen.

| **SacreBLEU-Score** | |
|---|---|
| **Modell** | **SacreBLEU** |
| BERT (original) | 82,01 |
| BERT (retrained) | 84,63 |

**Tabelle 8: Vergleich des nachtrainierten GECToR-BERT-Modells mit dem ursprünglichen Modell mit Hilfe des SacreBLEU-Scores.**

Zuletzt folgt nun die Evaluation durch die ROUGE-L-Metrik. Bei dieser Metrik konnten die Modelle keine signifikante Steigerung erreichen. Dies lässt sich jedoch dadurch erklären, dass in jedem Satz nur wenige Fehler injeziert wurden. Dadurch wird die LCS nur geringfügig verändert. Das führt zu hohen Scores, die sich jedoch nur schwach unterscheiden. Der F-Score der BERT-basierten Ausgangssystems erreichte einen F-Score von 92,04 (P = 92,67, R = 91,64). Das Nachtrainieren führte lediglich zu einer Steigerung von 1,32 Prozent auf F = 93,36 (P = 94,06, R = 92,81).

| **ROUGE-L** | | | |
|---|---|---|---|
| **Transformer-Modell** | **Precision** | **Recall** | **F-Score** |
| BERT (original) | 92,67 | 91,64 | 92,04 |
| BERT (retrained) | 94,06 | 92,81 | 93,36 |

**Tabelle 9: Vergleich des nachtrainierten GECToR-BERT-Modells mit dem ursprünglichen Modell mit Hilfe des ROUGE-L-Scores.**

Die einzelnen Metriken und die Steigerung, die durch das Retraining des Modells erreicht werden konnte, werden visuell in folgendem Diagramm dargestellt.

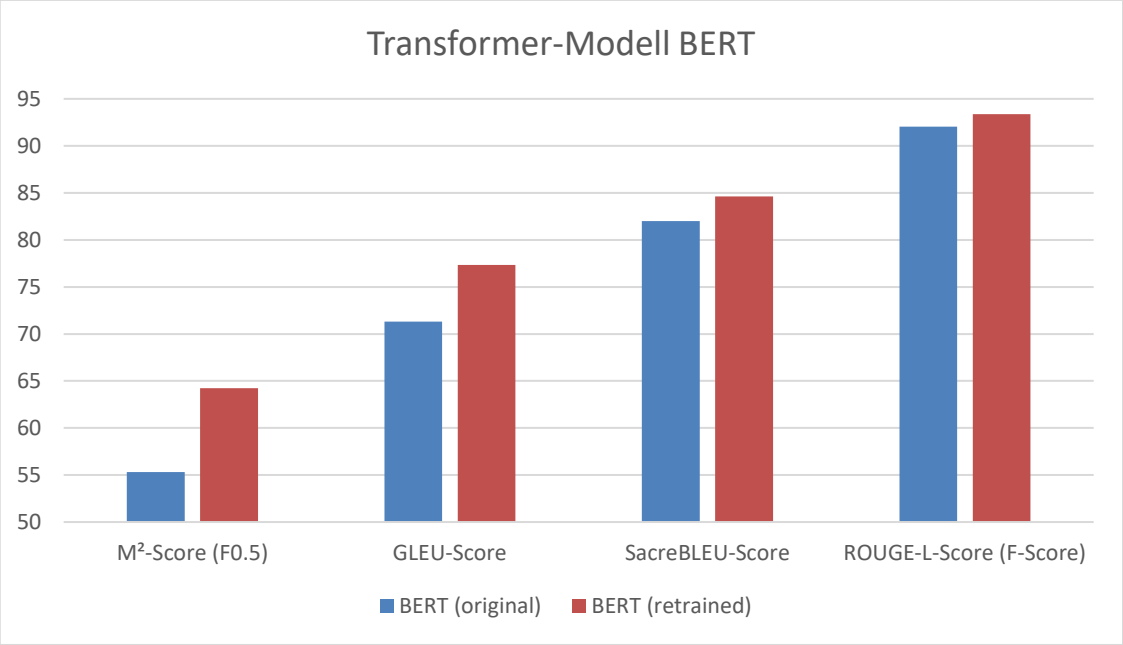


**Abbildung 21: Diagramm zum Vergleich des ursprünglichen BERT-Modells mit dem nachtrainierten Modell über alle verwendeten Metriken. (eigene Abbildung)**

Da das nachtrainierte Modell sich bei allen Metriken steigern konnte, kann davon ausgegangen werden, dass das Training mit einem Datensatz mit landwirtschaftlichen Inhalten und Vokabular sich positiv auf die Vorhersagen des Modells auswirken. Die Ergebnisse des Modells befinden sich bereits nach vier Epochen Training auf State-of-the-Art-Niveau.

**Transformer-Modell RoBERTa:**

Wie bei der Bewertung des RoBERTa-basierten Modells diente der F0.5-Score von 64,0 auf dem CoNNL-2014 Testdatensatz aus Ausgangsbasis für die Evaluation in dieser Arbeit. Auf dem in dieser Arbeit verwendeten Datensatz erreichte das Modell ein ähnliches Ergebnis. Der F-Score des Ausgangsmodells erreichte auf dem HORSCH Testdatensatz einen Wert von 64,57. Im Gegensatz zum BERT-basierten System sind die Ergebnisse des ursprünglichen Modells bereits auf State-of-the-Art-Niveau. Trotzdem konnte durch das Nachtrainieren auf einem themenspeziellen Datensatz die Leistung weiter steigern. Der F-Score nach vier Epochen Retraining erreichte einen Wert von 69,91, also einer Steigerung von 5,34. Genau wie bei BERT konnte die Performance des Modells durch eine enorme Steigerung des Recalls (+10,42) und durch eine Steigerung der Precision von +2,87 verbessert werden.

| **$M^2$-Score** | | | |
|---|---|---|---|
| **Transformer-Modell** | **Precision** | **Recall** | **F0.5** |
| RoBERTa (original) | 70,59 | 48,16 | 64,57 |
| RoBERTa (retrained) | 73,46 | 58,58 | 69,91 |

**Tabelle 10: Vergleich des nachtrainierten GECToR-RoBERTa-Modells mit dem ursprünglichen Modell mit Hilfe des $M^2$-Scores.**

Beim GLEU-Score war eine geringere Steigerung zwischen beiden Modellen zu beobachten. Der GLEU des vortrainierten Systems lag bei einem Wert von 75,63. Nach dem Training mit Hilfe des HORSCH-Datensatzes konnte der Score auf einen Wert von 80,10 gesteigert werden, was einer Verbesserung von 4,47 entspricht. Die Steigerung nach dieser geringen Trainingszeit kann jedoch immernoch als signifikant angesehen werden, da die Bewertung des Ausgangssystems mit einem F0.5 des $M^2$-Score schon sehr positiv ausfiel.

| **GLEU-Score** | |
|---|---|
| **Transformer-Modell** | **GLEU** |
| RoBERTa (original) | 75,63 |
| RoBERTa (retrained) | 80,10 |

**Tabelle 11: Vergleich des nachtrainierten GECToR-RoBERTa-Modells mit dem ursprünglichen Modell mit Hilfe des GLEU-Scores.**

Die Veränderungen der Werte des SacreBLEU- und ROUGE-L-Score fielen, ähnlich der BERT Evaluation, eher gering aus. Beim Sacre-BLEU konnte eine Steigerung von 2,28 Punkten erreicht werden. Beim ROUGE-L war eine Steigerung von 1,16 zu beobachten. Die geringen Änderungen sind ebenfalls auf die oben angeschsprochenen Faktoren zurückzuführen.

| **SacreBLEU-Score** | |
|---|---|
| **Modell** | **SacreBLEU** |
| RoBERTa (original) | 84,63 |
| RoBERTa (retrained) | 86,91 |

**Tabelle 12: Vergleich des nachtrainierten GECToR-RoBERTa-Modells mit dem ursprünglichen Modell mit Hilfe des SacreBLEU-Scores.**

Precision und Recall der ROUGE-L-Metrik konnten ebenfalls geringfügig gesteigert werden. Die Vohersagepräzision konnte um 0,94 von P = 94,23 auf P = 95,17 und die Fehlererkennungsrate um 1,35 von R = 92,66 auf R = 94,01 gesteigert werden.

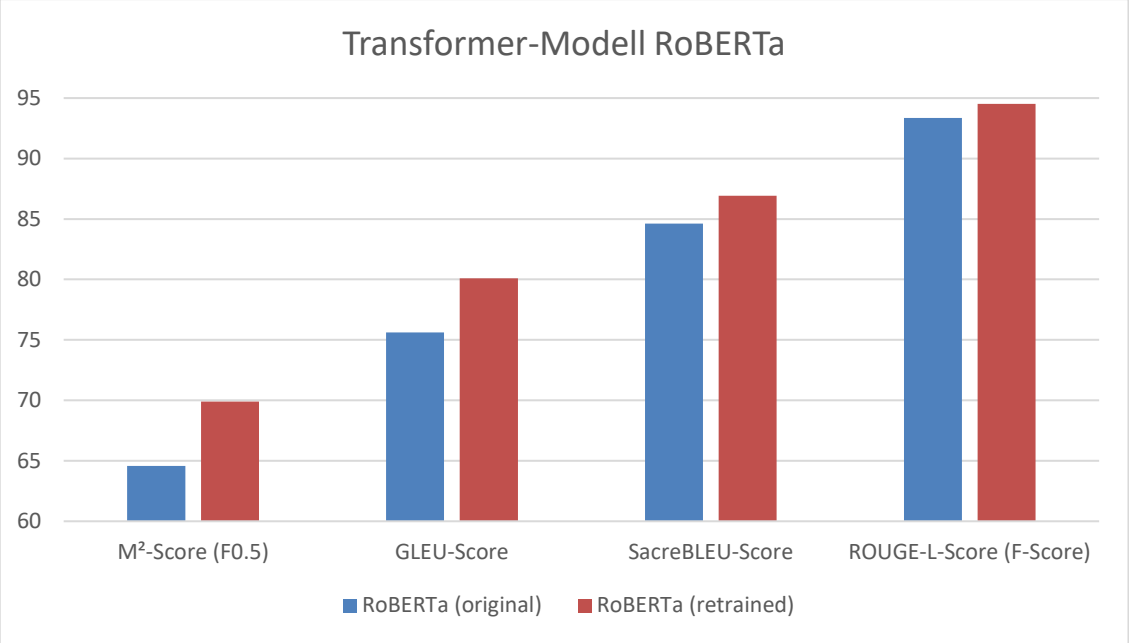


**Abbildung 22: Diagramm zum Vergleich des ursprünglichen RoBERTa-Modells mit dem nachtrainierten Modell über alle verwendeten Metriken. (eigene Abbildung)**

Da sich die Modelle im Gegensatz zu ihrem Referenzmodell ebenfalls in jeder Metrik steigern konnte, kann davon ausgegangen werden, dass sich die Modelle durch weiteres Nachtrainieren weiter steigern können. Das GECToR-Modell mit RoBERTa-Transformer konnte durch vier Epochen Training seine Leistung bereits über State-of-the-Art-Niveau hinaus steigern.

**Transformer-Modell XLNet:**

Das GECToR-System mit dem XLNet-Transformer lieferte in deren Test das beste Ergebnis auf dem CoNNL-2014 Datensatz mit einem F0.5 von 65,3. Damit war die Erwartung an diese Variante des Modells sehr hoch. In der Evaluation des Ausgangsmodells auf dem HORSCH Testdatensatz erreichte es jedoch nicht den Bestwert. Dieser lag mit 63,91 und einer Differenz von 0,66 knapp unter dem Ausgangsmodell mit dem RoBERTa-Transformer. Durch das Training auf dem Datensatz dieser Arbeit konnte jedoch der Bestwert für das GECToR-Modell erreicht werden. Der F0.5-Score erreichte einen Wert von 69.93. Herausragend dabei war der Recall der ebenfalls den Bestwert unter den GECToR-Modellen mit einem Wert von 59,35 aufstellte. Die Precision dieser Variante lag knapp unter dem Bestwert, der durch das RoBERTa-Modell gesetzt wurde, bei einem Wert von 73,18.

| **M²-Score** |
|---|

| **Transformer-Modell** | **Precision** | **Recall** | **F0.5** |
|---|---|---|---|
| XLNet (original) | 68,32 | 50,81 | 63,91 |
| XLNet (retrained) | 73,18 | 59,35 | 69,93 |

**Tabelle 13: Vergleich des nachtrainierten GECToR-XLNet-Modells mit dem ursprünglichen Modell mit Hilfe des $M^2$-Scores.**

Die Auswertung des GLEU-Scores verlief ähnlich positiv. Das Modell erreichte einen neuen Bestwert in meiner Evaluation mit GLEU = 80,86. Damit konnte es sich wiederholt gegen des RoBERTa-Transformer durchsetzen. Die Differenz zwischen XLNet und RoBERTa lag bei 0,76 Punkten.

| **GLEU-Score** | |
|---|---|
| **Transformer-Modell** | **GLEU** |
| XLNet (original) | 76,43 |
| XLNet (retrained) | 80,86 |

**Tabelle 14: Vergleich des nachtrainierten GECToR-XLNet-Modells mit dem ursprünglichen Modell mit Hilfe des GLEU-Scores.**

XLNet lieferte beim SacreBLEU-Score ebenfalls die besten Ergebnisse. Der Score des nachtrainierten Modells erreichte einen Wert von 87,16. Der Unterschied zum ursprünglichen Modell lag bei 2,44 Punkten.

| **SacreBLEU-Score** | |
|---|---|
| **Modell** | **SacreBLEU** |
| XLNet (original) | 84,72 |
| XLNet (retrained) | 87,16 |

**Tabelle 15: Vergleich des nachtrainierten GECToR-XLNet-Modells mit dem ursprünglichen Modell mit Hilfe des SacreBLEU-Scores.**

Bei der Evaluation durch die ROUGE-L-Metrik landete GECToR-XLNet knapp unter den Bestwerten der RoBERTa Variante. Erreicht wurde ein F-Score von 94,39 im Gegensatz zum Ausgangsmodell mit einem F-Score von 93,31. Dabei konnte die Vorhersagepräzision um 0,95 gesteigert werden (F(original) = 94,11, F(retrained) = 95,06). Der Recall konnte ebenfalls leicht gesteigert werden. Er erreichte einen Wert von 93,87. Das usrprüngliche System kam auf einen Wert von 92,7. Das entspricht einer Differenz von 1,17 Punkten.

| ROUGE-L | | | |
|---|---|---|---|
| **Transformer-Modell** | **Precision** | **Recall** | **F-Score** |
| XLNet (original) | 94,11 | 92,7 | 93,31 |
| XLNet (retrained) | 95,06 | 93,87 | 94,39 |

**Tabelle 16: Vergleich des nachtrainierten GECToR-XLNet-Modells mit dem ursprünglichen Modell mit Hilfe des ROUGE-L-Scores.**

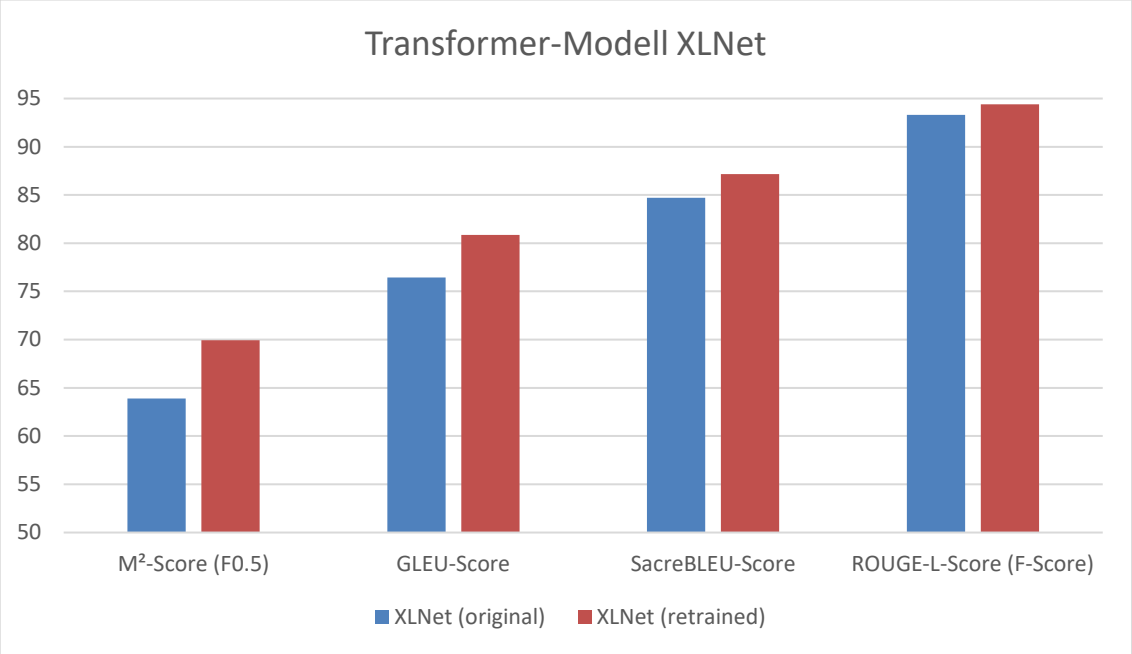


**Abbildung 23: Diagramm zum Vergleich des ursprünglichen XLNet-Modells mit dem nachtrainierten Modell über alle verwendeten Metriken. (eigene Abbildung)**

**Eigene Modelle:**

Als Referenz zu den genannten, nachtrainierten Modellen der GECToR-Architektur wurden zwei Modelle derselben Architektur von Grund auf neu traniert. Ein Modell wurde mit einem neuen Vokabular initialisiert und das andere Modell wurde mit dem bestehenden Vokabular trainiert, wie es bei den vortrainierten Modellen verwendet wurde. Die Trainingsdauer wurde ebenfalls auf vier Epochen gesetzt. Als Transformer-Modell wurde BERT verwendet. Dadurch sollte herausgefunden werden, wie sehr die Performance von den vortrainierten Modellen abhängt. Die Ergebnisse waren zum Teil sehr überraschend. Bei der Bewertung durch den MaxMatch-Score war die Präzision etwas schlechter als bei den vorher genannten Modellen, jedoch erreichte sie verhältnismäßig hohe Werte von 64,29 (HORSCH Vokabular) und 61,07 (ursprüngliches

Vokabular). Im Gegensatz zur Precision fiel der Recall sehr schlecht aus. Dieser erreichte Werte von nur 5,24 (HORSCH Vokabular) und 5,18 (ursprüngliches Vokabular). Der F-Score beider Modelle fiel aufgrund dessen mit F0.5 = 19,77 und F0.5 = 19,33 eher schlecht aus.

| **$M^2$-Score** | | | |
|---|---|---|---|
| **Transformer-Modell** | **Precision** | **Recall** | **F0.5** |
| BERT/HORSCH Vokabular | 64,29 | 5,24 | 19,77 |
| BERT/GECToR Vokabular | 61,07 | 5,18 | 19,33 |

**Tabelle 17: Vergleich der eigenen GECToR-Modelle mit dem ursprünglichen Modell mit Hilfe des $M^2$-Scores.**

Da die Ergebnisse der anderen Scores im Verhältnis zu den vortrainierten Modellen ähnlich schlecht ausfielen, werden diese hier nicht weiter erläutert. Die einzelnen Werte können jedoch in folgender Tabelle nachgeschlagen werden.

| **GLEU-Score** | | | |
|---|---|---|---|
| **Transformer-Modell** | **GLEU** | | |
| BERT/HORSCH Vokabular | 53,31 | | |
| BERT/GECToR Vokabular | 53,29 | | |
| **SacreBLEU-Score** | | | |
| **Modell** | **SacreBLEU** | | |
| BERT/HORSCH Vokabular | 76,71 | | |
| BERT/GECToR Vokabular | 76,67 | | |
| **ROUGE-L** | | | |
| **Transformer-Modell** | **Precision** | **Recall** | **F-Score** |
| BERT/HORSCH Vokabular | 78,55 | 76,59 | 77,45 |
| BERT/GECToR Vokabular | 78,49 | 76,55 | 77,39 |

**Tabelle 18: Vergleich der eigenen GECToR-Modelle mit dem ursprünglichen Modell mit Hilfe von GLEU, SacreBLEU und ROUGE-L.**

Nach der Evaluation der GECToR-Modelle folgt nun die Bewertung des PIE-Modells auf dem HORSCH-eigenen Testdatensatz. Strukturell werden die Ergebnisse an denselben Metriken ebenfalls textlich, tabellarisch und in Form von Diagrammen ausgewertet. Grundlage für das PIE-Modell und das Training für diese Arbeit war ein Modell, das mit oben beschriebenen Parametern nachtrainiert wurde.

PIE lieferte in seinem ursprünglichen Zustand Ergebnisse, die deutlich unter den Werten der GECToR-Modelle angesiedelt waren. Beim $M^2$-Score kam das vortrainierte Modell auf eine Precision von 29,77 und einen Recall von 33,66. Daraus ergibt sich ein F0.5 = 30,47. Nach dem Retraining konnten diese Werte auf P = 41,31, R = 60,45 und F0.5 = 44,10 gesteigert werden.

| **$M^2$-Score** | | | |
|---|---|---|---|
| **Transformer-Modell** | **Precision** | **Recall** | **F0.5** |
| PIE (original) | 29,77 | 33,66 | 30,47 |
| PIE (retrained) | 41,31 | 60,45 | 44,10 |

**Tabelle 19: Vergleich des nachtrainierten PIE-Modells mit dem ursprünglichen Modell mit Hilfe des $M^2$-Scores.**

Die Evaluation des GLEU- und SacreBLEU-Scores lieferte analoge Ergebnisse. Beide Metriken konnten mit dem vortrainierten Modell einen mittelmäßigen Score erreichen, konnten sich innerhalb dieser vier Epochen jedoch deutlich verbessern.

| **GLEU-Score** | |
|---|---|
| **Transformer-Modell** | **GLEU** |
| PIE (original) | 52,99 |
| PIE (retrained) | 52,54 |
| **SacreBLEU-Score** | |
| **Modell** | **SacreBLEU** |
| PIE (original) | 79,33 |
| PIE (retrained) | 84,63 |

**Tabelle 20: Vergleich des nachtrainierten PIE-Modells mit dem ursprünglichen Modell mit Hilfe des GLEU- und SacreBLEU-Scores.**

Beim ROUGE-L-Score lieferte PIE die schlechtesten Ergebnisse der gesamten Evaluation. So konnte das Modell selbst durch Retraining nicht über einen F-Score von 88,23 hinauskommen. Dieser F-Score setzt sich aus P = 88.12 und R = 88,54 zusammen.

| ROUGE-L | | | |
|---|---|---|---|
| **Transformer-Modell** | **Precision** | **Recall** | **F-Score** |
| PIE (original) | 86,31 | 85,66 | 85,84 |
| PIE (retrained) | 88,12 | 88,54 | 88,23 |

**Tabelle 21: Vergleich des nachtrainierten PIE-Modells mit dem ursprünglichen Modell mit Hilfe des ROUGE-L-Scores.**

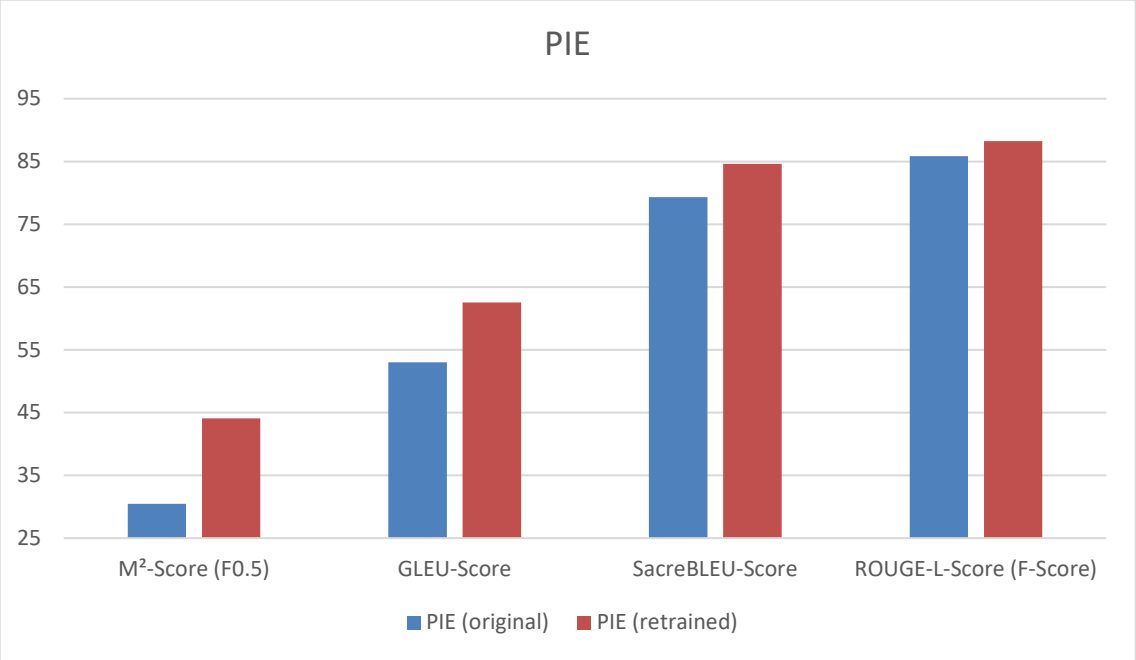


**Abbildung 24: Diagramm zum Vergleich des ursprünglichen PIE-Modells mit dem nachtrainierten Modell über alle verwendeten Metriken. (eigene Abbildung)**

Das letzte Modell, das in dieser Arbeit evaluiert wird, ist Fairseq-GEC. Als Grundlage diente, ähnlich wie bei PIE, ein Modell, welches im ursprünglichen Zustand mit dem nachtrainierten Modell evaluiert und verglichen wird.

Das Ausgangsmodell on Fairseq-GEC lieferte auf dem verwendeten Testdatensatz eher mittelmäßige Ergebnisse. Der F0.5-Score des vortrainierten Modells erreichte einen Score von 29,35, was ungefähr der Hälfte der GECToR-Modelle entspricht. Die Präzision des Modells erzielte einen Wert von 28,42. Beim Recall lag der Wert bei 33,79. Umso überraschender war es den M²-Score des nachtrainierten Modells zu sehen. Das Training

dieses Modells führte zur größten Steigerung in meinen Experimenten. Der F-Score steigerte sich um 24,08 Punkte und erreichte somit einen Wert von 53,43. Dies ist auf die Verbesserung der Pecision zurückzuführen. Mit einer Steigerung um 30,59 Punkten konnte ein Wert von 59,01 auf dem Testdatensatz evaluiert werden. Der Recall hingegen steigerte sich verhältnismäßig wenig. Mit einem Unterschied von 5,02 kam die Bewertung durch den $M^2$-Score auf einen Recall R = 38,77.

| **$M^2$-Score** | | | |
|---|---|---|---|
| **Transformer-Modell** | **Precision** | **Recall** | **F0.5** |
| Fairseq-GEC (original) | 28,42 | 33,79 | 29,35 |
| Fairseq-GEC (retrained) | 59,01 | 38,77 | 53,43 |

**Tabelle 22: Vergleich des nachtrainierten Fairseq-GEC-Modells mit dem ursprünglichen Modell mit Hilfe des $M^2$-Scores.**

GLEU- und SacreBLEU-Score konnten durch das Nachtrainieren eine Verbesserung von ca. 10% nachgewiesen werden. Die Modelle erreichten Werte von GLEU = 62,65 (ursprüngliches Modell) und GLEU = 70,39 (nachtrainiertes Modell). Der SacreBLEU-Score kamen die Modelle auf BLEU = 71,64 (ursprüngliches Modell) und BLEU = 81,07 (nachtrainiertes Modell).

| **GLEU-Score** | |
|---|---|
| **Transformer-Modell** | **GLEU** |
| Fairseq-GEC (original) | 62,65 |
| Fairseq-GEC (retrained) | 70,39 |
| **SacreBLEU-Score** | |
| **Modell** | **SacreBLEU** |
| Fairseq-GEC (original) | 71,64 |
| Fairseq-GEC (retrained) | 81,07 |

**Tabelle 23: Vergleich des nachtrainierten Fairseq-GEC-Modells mit dem ursprünglichen Modell mit Hilfe des GLEU- und SacreBLEU-Scores.**

Bei der Bewertung durch die Metrik ROUGE-L waren Veränderungen im Bereich von ca. 6% zu beobachten. Dabei kam das ursprüngliche Modell auf einen F-Score von 85,73

und das nachtrainierte Modell auf F = 91,72. Diese Veränderungen sind auf den Zuwachs von 6,1 Punkten bei der Präzision und der Steigerung des Recalls um 5,59 zurückzuführen. Alle entsprechenden Werte können der nachfolgenden Tabelle entnommen werden.

| **ROUGE-L** | | | |
|---|---|---|---|
| **Transformer-Modell** | **Precision** | **Recall** | **F-Score** |
| Fairseq-GEC (original) | 86,69 | 85,29 | 85,73 |
| Fairseq-GEC (retrained) | 92,79 | 90,88 | 92,72 |

**Tabelle 24: Vergleich des nachtrainierten Fairseq-GEC-Modells mit dem ursprünglichen Modell mit Hilfe des ROUGE-L-Scores.**

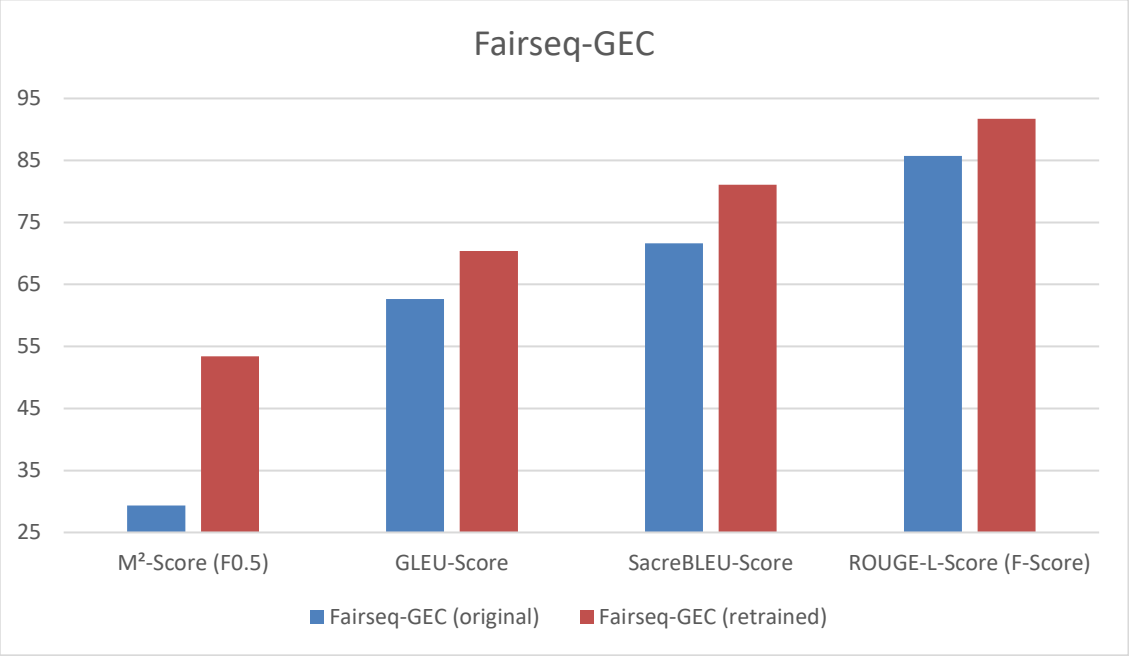


**Abbildung 25: Diagramm zum Vergleich des ursprünglichen Fairseq-GEC-Modells mit dem nachtrainierten Modell über alle verwendeten Metriken. (eigene Abbildung)**

Abschließend lässt sich festhalten, dass sich alle Modelle durch das Nachtrainieren mit einem fachspezifischen Datensatz verbessern konnten. Jedes der einzelnen Modelle konnte sich hinsichtlich der vier verwendeten Metriken signifikant steigern. Diese Erkenntnis ist vielversprechend, wenn es um das weitere Nachtrainieren dieser Modelle geht.

## 5.3 Auswahl der erfolgreichsten Modelle für weiteres Retraining

Den Abschluss der ersten Zwischenevaluation bildet die Auswahl der Modelle für das weitere Training. Das Auswahlverfahren basiert auf zwei Fragen:

- Welche(s) Modell(e) konnten die besten Ergebnisse bei den Metriken erreichen?
- Welche(s) Modell(e) könnte die größte Steigerung der Performance durch das Training erreichen?

Die Modelle mit den besten Ergebnissen sind, wie den Tabellen zu entnehmen ist, GECToR mit den Transformer-Modellen RoBERTa und XLNet. Diese beiden Modelle konnten die Bestwerte über alle Metriken hinweg für sich gewinnen. Deswegen werden diese beiden Modelle Teil des weiteren Trainings sein. Ebenfalls weiter verfolgt wird das GECToR-Modell mit dem BERT Transformer, da sich alle GECToR-Modelle zu Ensembles zusammenfassen lassen. Die Ensemblebildung soll nach dem weiteren Training stattfinden. Fairseq-GEC konnte bei der Zwischenevaluation durch sein enormes Steigerungspotential überzeugen. Deswegen wird auch das Training dieses Modells bis zur finalen Evaluation weiterverfolgt. PIE konnte in dieser Evaluation nach vier Epochen Training keine Top-Werte erzielen und schaffte es auch nicht sich so sehr zu steigern, wie es bei Fairseq-GEC der Fall war. Aufgrund der hohen Trainingszeit und den Kosten, die mit dem Training von PIE verbunden waren, wird dieser Ansatz für die finale Evaluation nicht weiterverfolgt. PIE war das einzige Modell, das für meine privaten Ressourcen zu groß und umfangreich war. Deshalb musste es auf einer GPU von AWS trainiert werden und verursachte Kosten in Höhe von knapp 100€.

# 6 Finale Evaluation

Im folgenden Kapitel werden die Ergebnisse der finalen Evaluation vorgestellt. Im Anschluss an die Zwischenevaluation nach vier Epochen wurden, die in Kapitel 5.3 ausgewählten Modelle noch für weitere 16 Epochen nachtrainiert und das beste Modell aus den insgesamt 20 Epochen ausgewählt. Die Auswertung findet auf demselben Testdatensatz statt, der auch schon für die vorherige Bewertung verwendet wurde. Es wird wieder anhand der in Kapitel 5.1 vorgestellten Metriken bewertet. Die Modelle sollen nun gegenübergestellt werden, um am Ende das beste Modell auszuwählen.

## 6.1 Vorstellung der Ergebnisse

In dieser Sektion sollen nun die Ergebnisse aller Modelle über alle oben genannten Metriken gegenübergestellt werden.

| **$M^2$-Score** | | | |
|---|---|---|---|
| **Modell** | **Precision** | **Recall** | **F0.5** |
| GECToR-BERT | 66,76 | 64,72 | 66,34 |
| GECToR-RoBERTa | **72,91** | 58,71 | 69,54 |
| GECToR-XLNET | 69,97 | **69,84** | **69,95** |
| Fairseq-GEC | 56,66 | 39,94 | 52,28 |

**Tabelle 25: Vergleich der verschiedenen Modellarchitekturen anhand des besten Modells aus 20 Epochen Nachtrainieren. Evaluation mittels $M^2$-Score.**

Die Modelle der GECToR-Architektur schneiden bei der Evaluation nach 20 Epochen am besten ab. Das Modell von Fairseq-GEC liegt mit einer Differenz im F05 von -14,06 Punkten weit hinter dem schlechtesten GECToR-Modell. Die Modelle dieser Architektur schneiden bei der Auswertung des MaxMatch-Scores im Allgemeinen sehr gut ab. Das beste Modell ist da Modell, das den XLNet-Transformer verwendet. Die Precision des XLNet-Modells liegt zwar mit einem Wert von 69,97 knapp hinter dem des Modells mit RoBERTa-Transformer, kann jedoch mit dem Bestwert im Recall von 69,84 punkten. Daraus ergibt sich ein F0.5 von 69,95.

| **Modell** | **GLEU** |
|---|---|
| GECToR-BERT | 81,37 |
| GECToR-RoBERTa | 80,09 |

| | |
|---|---|
| GECToR-XLNET | **83,64** |
| Fairseq-GEC | 70,31 |

**Tabelle 26: Vergleich der verschiedenen Modellarchitekturen anhand des besten Modells aus 20 Epochen Nachtrainieren. Evaluation mittels GLEU-Score.**

Beim GLEU-Score schneiden alle betrachteten Modelle ähnlich gut ab. Den Bestwert erreicht erneut das GECToR-Modell mit XLNet-Transformer. Mit einem Wert von 83,64 kann das Modell einen Abstand zum nächstbesten Modell von 2,27 Punkten aufbauen. Das schlechteste System ist erneut das Modell von Fairseq-GEC, welches einen Score von 70,31 erreicht.

| **Modell** | **SacreBLEU** |
|---|---|
| GECToR-BERT | 86,04 |
| GECToR-RoBERTa | 86,71 |
| GECToR-XLNET | **87,57** |
| Fairseq-GEC | 80,4 |

**Tabelle 27: Vergleich der verschiedenen Modellarchitekturen anhand des besten Modells aus 20 Epochen Nachtrainieren. Evaluation mittels SacreBLEU-Score.**

Die GECToR-Architektur kann nach einem Retraining von 20 Epochen bei der Bewertung durch den SacreBLEU-Score erneut Bestwerte erreichen. Die Verwendung des XLNet-Transformers zeigt sich hier ebenfalls vorteilhaft und erreicht einen Spitzenwert von 87,57. Die Ergebnisse der anderen GECToR-Modelle fallen jedoch ähnlich gut aus.

| **ROUGE-L** | | | |
|---|---|---|---|
| **Modell** | **Precision** | **Recall** | **F0.5** |
| GECToR-BERT | 94,03 | 93,34 | 93,61 |
| GECToR-RoBERTa | **95,17** | 93,84 | 94,35 |
| GECToR-XLNET | 95,04 | **94,34** | **94,62** |
| Fairseq-GEC | 92,54 | 90,24 | 91,26 |

**Tabelle 28: Vergleich der verschiedenen Modellarchitekturen anhand des besten Modells aus 20 Epochen Nachtrainieren. Evaluation mittels ROUGE-L-Score.**

Das Ergebnis des ROUGE-L auf den Vorhersagen der Modelle fällt ähnlich zu den vorherigen Metriken aus. Während die GECToR-Modelle die besten Werte erzielen konnten, liegt Fairseq-GEC etwas zurück. Den Bestwert stellt erneut GECToR mit XLNet-Transformer mit einem F-Score von 94,62.

Der Vergleich der Modelle über alle Metriken kann dem untenstehenden Diagramm entnommen werden.

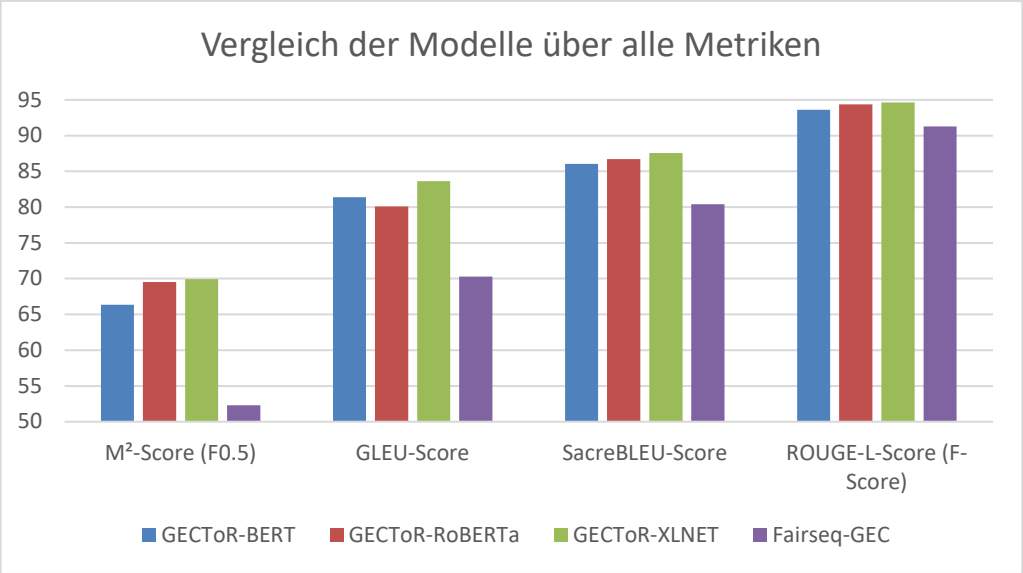


**Abbildung 26: Vergleich aller vollständig nachtrainierten Modelle untereinander zur Bestimmung des erfolgreichsten Modells. (eigene Abbildung)**

## 6.2 Ergebnisse der Ensemblebildung der GECToR-Modelle und Vergleich mit den Bestwerten der einzelnen Modelle

In den Experimenten wurden zwei Ensembles gebildet. Das erste Ensemble ist eine Verbindung der beiden stärksten GECToR-Modelle. Es handelt sich um den Zusammenschluss von den Systemen mit RoBERTa- und XLNet-Transformer. Der zweite Zusammenschluss umfasst alle trainierten GECToR-Modelle.

| **M²-Score** | | | |
|---|---|---|---|
| **Modell** | **Precision** | **Recall** | **F0.5** |
| Ensemble RoBERTa-XLNet | **76,61** | 65,50 | 74,10 |
| Ensemble BERT-RoBERTa-XLNet | 76,37 | 66,37 | **74,23** |

**Tabelle 29: Vorstellung der Ergebnisse der Ensemblebildung der GECToR-Modelle gemessen am M²-Score.**

Allgemein lässt sich sagen, dass die Ensemblebildung die Performance der Systeme erneut enorm steigern konnte. Die Leistung der Ensembles, gemessen am M²-Score, konnte die Precision stark verbessern. Lediglich der Recall konnte im Gegensatz zu den einzelnen Modellen nicht gesteigert werden. Die beste Precision erreichte das Ensemble

aus RoBERTa- und XLNet-Transformer mit einem Wert von 76,61. Mit einem Recall von 66,37 erreichte jedoch der Zusammenschluss aus allen drei Modellen den besten Wert. Aus den Werten von Precision und Recall errechnet sich ein F0.5 = 74,23 des Kollektivs aus allen drei Systemen.

| **Modell** | **GLEU** | **SacreBLEU** |
|---|---|---|
| Ensemble RoBERTa-XLNet | 83,68 | 88,64 |
| Ensemble BERT-RoBERTa-XLNet | **84,17** | **88,87** |

**Tabelle 30: Vorstellung der Ergebnisse der Ensemblebildung der GECToR-Modelle gemessen am GLEU- und SacreBLEU-Score.**

Die Unterschiede der Bewertung durch GLEU und SacreBLEU fallen ähnlich gering aus wie die des MaxMatch-Scores. Das Ensemble aus BERT, RoBERTa und XLNet konnte leicht bessere Ergebnisse erzielen.

Diese Ergebnisse lassen sich auch auf die ROUGE-L-Metrik übertragen, bei der beide Modelle ähnliche Werte erzielen können.

| **ROUGE-L** | | | |
|---|---|---|---|
| **Modell** | **Precision** | **Recall** | **F-Score** |
| Ensemble RoBERTa-XLNet | **95,9** | 84,54 | 95,16 |
| Ensemble BERT-RoBERTa-XLNet | 95,8 | **94,66** | **95,18** |

**Tabelle 31: Vorstellung der Ergebnisse der Ensemblebildung der GECToR-Modelle gemessen am ROUGE-L-Score.**

Im Vergleich zu den einzelnen Modellen konnten durch die Ensemblebildung wesentliche Verbesserungen erzielt werden. Im Gegensatz zu den einzelnen Systemen konnten Verbesserungen von bis zu 4% erzielt werden. In nahezu allen Metriken konnten die Ensembles bessere Leistungen zeigen.

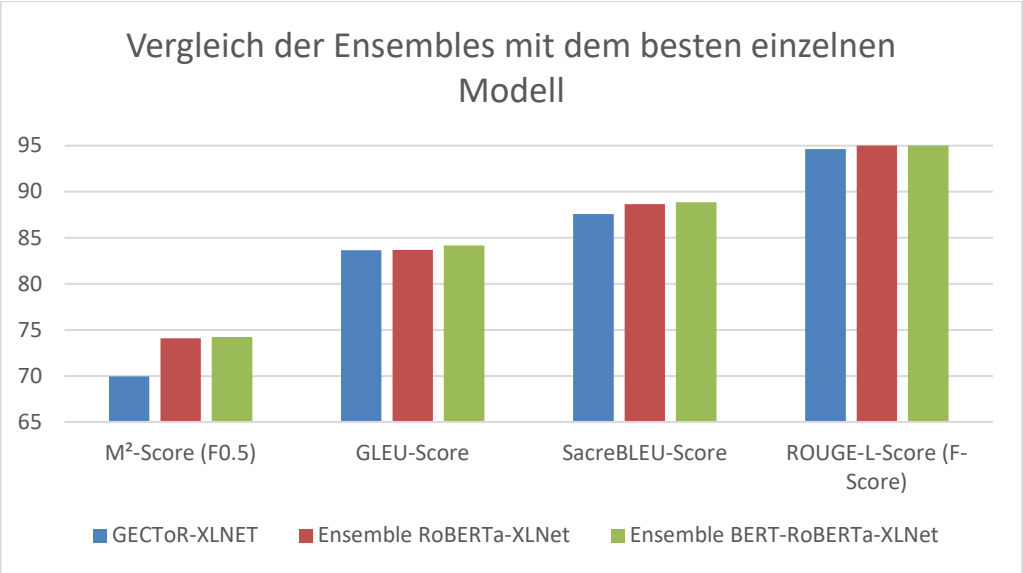


**Abbildung 27: Vergleich der gebildeten Ensembles aus den vollständig nachtrainierten GECToR-Modellen mit dem erfolgreichsten einzelnen Modell. (eigene Abbildung)**

## 6.3 Schlussfolgerungen und Interpretation

Aus der Evaluation lässt sich schlussfolgern, dass das Nachtrainieren auf dem Datensatz der landwirtschaftlichen Domäne positiven Einfluss auf die Performance der Modelle in der gegebenen Problemstellung hat. Die ersten Epochen des Retraining zeigten sich effektiver als die nachfolgenden 16 Epochen. Vergleicht man die Werte des M²-Score der ersten vier Epochen mit den Ergebnissen der finalen Evaluation kann man feststellen, dass sich in der ersten Phase des Nachtrainierens sowohl Precision als auch Recall enorm verbessern konnten. Bei der zweiten Phase des Retrainings konnte lediglich der Recall gesteigert werden. Die Evaluationsergebnisse zur Precision konnten keinen positiven Effekt aus der zweiten Trainingsphase feststellen. In manchen Fällen sank die Precision mininmal. Wie Omelianchuk et al. im GitHub-Repository zu ihrer Arbeit beschreiben resultiert dieses Phänomen aus der Empfindlichkeit dieser Modelle gegenüber random seeds, Modellinitialisierung, der Datenreihenfolge, etc. Daraus folgt, dass mit zunehmender Länge des Trainings der Recall immer weiter verbessert werden kann, dies jedoch auf Kosten der Precision geschieht. Aus diesem Grund ist es wichtig, das Training zum richtigen Zeitpunkt abzubrechen. (Omelianchuk et al., 2020)

Wählt man das beste Modell aus den nachtrainierten Modellen aus, fällt die Wahl für ein einzelnes Modell auf das GECToR-System mit XLNet-Transformer. Dieses Modell konnte die besten Ergebnisse erzielen, wenn man die Ensembles außen vorlässt.

Betrachtet man die Ensembles fällt die Wahl klar auf das Ensemble der drei GECToR-Modelle, da sie bei nahezu allen Metriken die beste Performance erzielen konnten.

# 7 Zusammenfassung und Ausblick

In dieser Arbeit wurden drei unterschiedliche Architekturen von GEC-System mit Hilfe eines eigenen domänenspezifischen Datensatzes nachtrainiert. Zuerst wurde in einer Zwischenevaluation untersucht, ob sich das Retraining der Modelle positiv auf die Performance auswirken kann. Nach der ersten Auswertung folgte ein weiterer Trainingszeitraum bis insgesamt 20 Epochen Training abgeschlossen waren. Aus den zwischengespeicherten Modellen wurde für jedes Modell das beste ausgewählt. Anschließend folgte der Vergleich der Systeme untereinander. Als Modelle wurden GECToR, PIE und Fairseq-GEC ausgewählt. Für PIE zeigte sich schon bei der Zwischenevaluation, dass es auf der gegebenen Problemstellung keine guten Ergebnisse liefern kann. Die Steigerung während des Retrainings war ebenfalls schlechter als bei anderen Modellen. Fairseq-GEC zeigte bei der ersten Bewertung die größte Leistungssteigerung unter allen untersuchten Modellen. An diesen Anstieg konnte das Modell in den 16 Folgeepochen nicht anknüpfen. Die Architektur von GECToR unterscheidet zwischen drei verschiedenen Modellen mit verschiedenen Transformer-Modellen (BERT, RoBERTa, XLNet). Alle drei Modelle lieferten bereits in der Zwischenauswertung sehr gute Ergebnisse. Die weiteren Epochen ergaben zwar keine signifikanten Steigerungen mit sich, doch die GECToR-Architektur konnte seine Performance durch Ensemblebildung weiter steigern. Damit ist das Ensemble aller drei Modelle, mit der besten Leistung auf dem Testdatensatz, das am besten geeignete Modell für den Einsatz auf der geplanten Plattform der Firma HORSCH GmbH.

Limitiert wird das Nachtrainieren der oben genannten Modelle durch den Einsatz eines Datensatzes, in den synthetische Fehler injiziert wurden. Dadurch kann nicht sichergestellt werden, dass der Datensatz das Realweltproblem optimal abbilden kann. So könnte durch Training mit Daten aus realen Situationen möglicherweise bessere Ergebnisse erzielt werden.

Eine weitere Limitierung dieser Arbeit spiegelt sich in der Tatsache wider, dass der Trainings- und Testdatensatz aus zwei unterschiedlichen Quellen stammen. Auf der einen

Seite sind die Sätze des Testdatensatzes komplexer und stellen das getestete System vor größere Herausforderungen. Auf der anderen Seite kann der strukturelle Unterschied der Sätze die Bewertung geringfügig verzerren.

Für weitere Experimente könnte ein Datensatz aus der Realwelt mit organischen Fehlern erhoben werden. Damit könnte festgestellt werden, welche Auswirkungen das Training mit synthetischen Daten auf die Performance und die Ergebnisse hat. Außerdem kann somit sichergestellt werden, dass das gegebene Problem ausreichend auf das Modell abgebildet wird.

Außerdem könnte untersucht werden, wie die nachtrainierten Modelle auf offiziellen Datensätzen wie dem CoNLL-2014 Shared Task Datensatz, JFLEG Datensatz oder BEA Shared Task – 2019 abschneiden, um zu erforschen, ob sich die Modelle weiter steigern konnten und ob sie an den State-of-the-Art anknüpfen können.

## Literaturverzeichnis


Applying Unsupervised Machine Learning to Sequence Labeling. (2020, 22. Februar). Medium. https://medium.com/mosaix/deep-text-representation-for-sequence-labeling-2f2e605ed9d

Awasthi, A., Sarawagi, S., Goyal, R., Ghosh, S., & Piratla, V. (2019). Parallel Iterative Edit Models for Local Sequence Transduction. In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP) (pp. 4260–4270). Association for Computational Linguistics.

Ba, J. L., Kiros, J. R., & Hinton, G. E. (2016). Layer normalization. arXiv preprint arXiv:1607.06450.

Chen, M., Firat, O., Bapna, A., Johnson, M., Macherey, W., Foster, G., Jones, L., Schuster, M., Shazeer, N., Parmar, N., Vaswani, A., Uszkoreit, J., Kaiser, L., Chen, Z., Wu, Y., & Hughes, M. (2018). The Best of Both Worlds: Combining Recent Advances in Neural Machine Translation. In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers) (pp. 76–86). Association for Computational Linguistics.

Chollampatt, S., & Ng, H. (2018). A Multilayer Convolutional Encoder-Decoder Neural Network for Grammatical Error Correction. In Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence.

Dahlmeier, D., & Ng, H. (2012). Better Evaluation for Grammatical Error Correction. In Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (pp. 568–572). Association for Computational Linguistics.

DeepAI. (2020, 25. Juni). Neural Machine Translation. https://deepai.org/machine-learning-glossary-and-terms/neural-machine-translation

Grammatische Theorien. (2020, 18. April). ZUM Deutsch Lernen. https://deutsch-lernen.zum.de/wiki/Grammatische_Theorien#Valenz-.2FDependenzgramatik

Horev, R. H. (2018, 17. November). BERT Explained: State of the art language model for NLP. Medium. https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270

Informatik, G. F. (2017, 2. März). Deep Learning. Gesellschaft für Informatik e.V. (GI). https://gi.de/informatiklexikon/deep-learning#:%7E:text=3%20Vanishing%20Gradient%20Problem&text=Bei%20Verwendung%20von%20Aktivierungsfunktionen%20mit,und%20das%20Netz%20nicht%20konvergiert

Ji, J., Wang, Q., Toutanova, K., Gong, Y., Truong, S., & Gao, J. (2017). A Nested Attention Neural Hybrid Model for Grammatical Error Correction. In Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers) (pp. 753–762). Association for Computational Linguistics.

K. He, X. Zhang, S. Ren, & J. Sun (2016). Deep Residual Learning for Image Recognition. In 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (pp. 770-778).

Lin, C.Y. (2004). ROUGE: A Package for Automatic Evaluation of Summaries. In Text Summarization Branches Out (pp. 74–81). Association for Computational Linguistics.

Lin, C.Y., & Och, F. (2004). Automatic Evaluation of Machine Translation Quality Using Longest Common Subsequence and Skip-Bigram Statistics. In Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL-04) (pp. 605–612).

Luong, T., Pham, H., & Manning, C. (2015). Effective Approaches to Attention-based Neural Machine Translation. In Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (pp. 1412–1421). Association for Computational Linguistics.

Malmi, E., Krause, S., Rothe, S., Mirylenka, D., & Severyn, A. (2019). Encode, Tag, Realize: High-Precision Text Editing. In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP) (pp. 5054–5065). Association for Computational Linguistics.

Marek Rei, & Helen Yannakoudakis (2016). Compositional Sequence Labeling Models for Error Detection in Learner Writing. In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics.

Omelianchuk, K., Atrasevych, V., Chernodub, A., & Skurzhanskyi, O. (2020). GECToR – Grammatical Error Correction: Tag, Not Rewrite. In Proceedings of the Fifteenth Workshop on Innovative Use of NLP for Building Educational Applications (pp. 163–170). Association for Computational Linguistics.

Osborne, T., & Gerdes, K. (2019). The status of function words in dependency grammar: A critique of Universal Dependencies (UD). Glossa: a journal of general linguistics.

Papineni, K., Roukos, S., Ward, T., & Zhu, W.J. (2002). Bleu: a Method for Automatic Evaluation of Machine Translation. In Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (pp. 311–318). Association for Computational Linguistics.

Post, M. (2018). A Call for Clarity in Reporting BLEU Scores. In Proceedings of the Third Conference on Machine Translation: Research Papers (pp. 186–191). Association for Computational Linguistics.

Presseinformationen. (o. D.). DeepL. Abgerufen am 17. April 2021, von https://www.deepl.com/press.html

Rei, M., & Sogaard, A. (2018). Zero-Shot Sequence Labeling: Transferring Knowledge from Sentences to Tokens. In Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers) (pp. 293–302). Association for Computational Linguistics.

Rei, M., Crichton, G., & Pyysalo, S. (2016). Attending to Characters in Neural Sequence Labeling Models. In Proceedings of COLING 2016, the 26th International

Conference on Computational Linguistics: Technical Papers (pp. 309–318). The COLING 2016 Organizing Committee.

Ruder, S. R. (o. D.). Grammatical Error Correction. NLP-Progress. Abgerufen am 17. April 2021, von http://nlpprogress.com/english/grammatical_error_correction.html

Sagar Ailani, A. Dalvi, & Irfan Siddavatam (2019). Grammatical Error Correction (GEC): Research Approaches till now. International Journal of Computer Applications, 178, 1-3.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A., Kaiser, L., & Polosukhin, I. (2017). Attention is All you Need. In Advances in Neural Information Processing Systems. Curran Associates, Inc..

Vipul Raheja, & Dimitrios Alikaniotis (2020). Adversarial Grammatical Error Correction. ArXiv, abs/2010.02407.

Y. Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, M. Lewis, Luke Zettlemoyer, & Veselin Stoyanov (2019). RoBERTa: A Robustly Optimized BERT Pretraining Approach. ArXiv, abs/1907.11692.

Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R., & Le, Q. (2019). XLNet: Generalized Autoregressive Pretraining for Language Understanding. In Advances in Neural Information Processing Systems. Curran Associates, Inc..

Yonghui Wu and Mike Schuster and Zhifeng Chen and Quoc V. Le and Mohammad Norouzi and Wolfgang Macherey and Maxim Krikun and Yuan Cao and Qin Gao and Klaus Macherey and Jeff Klingner and Apurva Shah and Melvin Johnson and Xiaobing Liu and Lukasz Kaiser and Stephan Gouws and Yoshikiyo Kato and Taku Kudo and Hideto Kazawa and Keith Stevens and George Kurian and Nishant Patil and Wei Wang and Cliff Young and Jason Smith and Jason Riesa and Alex Rudnick and Oriol Vinyals and Greg Corrado and Macduff Hughes and Jeffrey Dean (2016). Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. CoRR, abs/1609.08144.

Zhao, W., Wang, L., Shen, K., Jia, R., & Liu, J. (2019). Improving Grammatical Error Correction via Pre-Training a Copy-Augmented Architecture with Unlabeled Data. arXiv preprint arXiv:1903.00138.

## Erklärung zur Urheberschaft

Ich habe die Arbeit selbständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt, sowie alle Zitate und Übernahmen von fremden Aussagen kenntlich gemacht.

Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt.

Die vorgelegten Druckexemplare und die vorgelegte digitale Version sind identisch.

| | |
|---|---|
| Ort, Datum | Unterschrift |

## Erklärung zur Lizenzierung und Publikation dieser Arbeit

**Name:** Christoph Meyer

**Titel der Arbeit:** Entwicklung einer Syntaxkorrektur im sprachlichen Kontext der Firma HORSCH GmbH

In der Regel räumen Sie mit Abgabe der Arbeit dem Lehrstuhl für Medieninformatik nur zwingend das Recht ein, dass die Arbeit zur Bewertung gelesen, gespeichert und vervielfältigt werden darf. Idealerweise liefern Seminararbeiten, Projektdokumentationen und Abschlussarbeiten aber einen Erkenntnisgewinn, von dem auch andere profitieren können. Wir möchten Sie deshalb bitten, uns weitere Rechte einzuräumen, bzw. idealerweise Ihre Arbeit unter eine freie Lizenz zu stellen.

Die in unseren Augen praktikabelsten Lösungen sind vorselektiert.

Hiermit gestatte ich die Verwendung der **schriftlichen Ausarbeitung** zeitlich unbegrenzt und nicht-exklusiv unter folgenden Bedingungen:

- ☐ Nur zur Bewertung dieser Arbeit
- ☐ Nur innerhalb des Lehrstuhls im Rahmen von Forschung und Lehre
- ☒ Unter einer Creative-Commons-Lizenz mit den folgenden Einschränkungen:
  - ☒ BY – Namensnennung des Autors
  - ☐ NC – Nichtkommerziell
  - ☐ SA – Share-Alike, d.h. alle Änderungen müssen unter die gleiche Lizenz gestellt werden.

(An Zitaten und Abbildungen aus fremden Quellen werden keine weiteren Rechte eingeräumt.)

Außerdem gestatte ich die Verwendung des im Rahmen dieser Arbeit erstellen **Quellcodes** unter folgender Lizenz:

- ☐ Nur zur Bewertung dieser Arbeit
- ☐ Nur innerhalb des Lehrstuhls im Rahmen von Forschung und Lehre
- ☐ Unter der CC-0-Lizenz (= beliebige Nutzung)
- ☒ Unter der MIT-Lizenz (= Namensnennung)
- ☐ Unter der GPLv3-Lizenz (oder neuere Versionen)

(An explizit mit einer anderen Lizenz gekennzeichneten Bibliotheken und Daten werden keine weiteren Rechte eingeräumt.)

Ich willige ein, dass der Lehrstuhl für Medieninformatik diese Arbeit – falls sie besonders gut ausfällt - auf dem Publikationsserver der Universität Regensburg veröffentlichen lässt.

Ich übertrage deshalb der Universität Regensburg das Recht, die Arbeit elektronisch zu speichern und in Datennetzen öffentlich zugänglich zu machen. Ich übertrage der Universität Regensburg ferner das Recht zur Konvertierung zum Zwecke der Langzeitarchivierung unter Beachtung der Bewahrung des Inhalts (die Originalarchivierung bleibt erhalten).

Ich erkläre außerdem, dass von mir die urheber- und lizenzrechtliche Seite (Copyright) geklärt wurde und Rechte Dritter der Publikation nicht entgegenstehen.

☒ Ja, für die komplette Arbeit inklusive Anhang

☐ Ja, für eine um vertrauliche Informationen gekürzte Variante (auf dem Datenträger beigefügt)

☐ Nein

☐ Sperrvermerk bis (Datum):

Regensburg, X

Ort, Datum

Unterschrift

## Inhalt des beigefügten Datenträgers

Beispiel (Ordner + Beschreibung):

| | |
|---|---|
| /1_Ausarbeitung | Die schriftliche Ausarbeitung als PDF und DOC |
| /2_Code | Quellcode und kompilierte Anwendung des Prototypen |
| /3_Studie/Design | Fragebogen und Script für die Benutzerstudie |
| /3_Studie/Rohdaten | Rohdaten der Studie im CSV-Format, inkl. Beschreibung der Felder |
| /4_Quellen | Alle in der Arbeit zitierten Quellen im PDF-Format |
| /5_Bilder | Alle selbst erstellten und aus anderen Quellen übernommenen Bilder |
| /6_Vorträge | Folien von Antritts- und Abschlussvortrag im PDF-Format |
| /7_Sonstiges | Notizen aus Besprechungen, Gedanken, … |

[Datenträger (CD, SD-Karte, o.ä.) hier oder auf Umschlaginnenseite einkleben]