Karlsruher Institut für Technologie

Software Design and Quality

Feinanpassung vs. Prompting: Der Fall der Wiederherstellung von Nachverfolgbarkeitsverbindungen

Bachelorarbeit von

Markus Bodenberger

An der KIT-Fakultät für Informatik
KASTEL – Institut für Informationssicherheit und Verlässlichkeit

1. Prüfer/Prüferin:	Prof. Dr.-Ing. Anne Koziolek
2. Prüfer/Prüferin:	Prof. Dr. Ralf Reussner
1. Betreuer/Betreuerin:	Dr.-Ing. Tobias Hey
2. Betreuer/Betreuerin:	M.Sc. Dominik Fuchß

30. Juni 2025 – 30. Oktober 2025

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

Feinanpassung vs. Prompting: Der Fall der Wiederherstellung von Nachverfolgbarkeitsverbindungen (Bachelorarbeit)

Ich versichere wahrheitsgemäß, die Arbeit selbständig verfasst, alle benutzten Quellen und Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde sowie die Satzung des KIT zur Sicherung guter wissenschaftlicher Praxis in der jeweils gültigen Fassung beachtet zu haben.

Karlsruhe, 30. Oktober 2025

. . . . . .

(Markus Bodenberger)

Zusammenfassung

Bei der Softwareentwicklung entstehen verschiedene Anforderungen, die zueinander in Beziehung stehen können. Diese Beziehungen werden mit Nachverfolgbarkeitsverbindungen (engl.: *trace links*, TLs) repräsentiert, die verschiedene Vorteile bieten, jedoch häufig gar nicht oder inkonsistent erfasst werden. Deshalb beschäftigt sich die Forschung mit der Wiederherstellung von TLs zwischen Anforderungen, für die es verschiedenste Ansätze gibt. Neuere automatisierte Ansätze verwenden häufig Feinanpassung oder Prompting, da sich damit oft bessere Ergebnisse erzielen lassen. Aktuell ist jedoch unklar, welche Feinanpassungs- und Prompting-Ansätze in welchen Anwendungsfällen die beste Performance liefern, da bislang kein umfassender Vergleich zwischen ihnen unternommen wurde.

In dieser Bachelorarbeit wird daher ein solcher Vergleich durchgeführt, um zu ermitteln, mit welcher Anzahl an verfügbaren Projekt-TLs welcher vollautomatisierte Feinanpassungs- und Prompting-Ansatz in welcher Situation die beste Performance erzielt. Dazu werden vier Szenarien betrachtet, in denen die Generierung und Vervollständigung von TLs mit und ohne die Nutzung von TLs aus anderen Projekten durchgeführt wird.

In Anwendungsfällen, in denen keine Projekt-TLs vorliegen und alle TLs generiert werden müssen, liefert ein aktueller *zero-shot* Prompting-Ansatz die beste Performance, gemessen an den F_1-Werten. Sind einige Projekt-TLs vorhanden, die vervollständigt werden müssen, dann erzielt der gleiche *zero-shot* Prompting-Ansatz bei wenigen vorhandenen TLs die besten F_1-Werte. Wenn keine anderen Projekte mit TLs verfügbar sind, dann wird im Durchschnitt ab ca. 45 vorhandenen Projekt-TLs Feinanpassung zum besten Ansatz, gemessen an den F_1-Werten. Liegen jedoch andere Projekte vor, dann ist dies schon ab etwa 20 TLs der Fall.

Die Ergebnisse dieser Arbeit liefern Praktikern, die sich mit der Wiederherstellung von TLs befassen, eine Orientierungshilfe bei der Auswahl des leistungsfähigsten Ansatzes und geben Forschern zugleich Hinweise auf mögliche Erweiterungen/Verbesserungen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

Commit Quelltext-Änderung (*commit*)

DA Datenaufteilung

FA Feinanpassungsansatz

FN Falsch negativ (*false negative*)

FP Falsch positiv (*false positive*)

HLR High-Level-Anforderung (*high-level requirement*)

IR Information Retrieval

Issue Fehlerbericht oder Feature-Anfrage (*issue*)

LLR Low-Level-Anforderung (*low-level requirement*)

ML Maschinelles Lernen (*machine learning*)

P Präzision (*precision*)

PA1 Prompting-Ansatz 1

PA2 Prompting-Ansatz 2

PA3 Prompting-Ansatz 3

PLM Vortrainiertes Sprachmodell (*pretrained language model*)

R Ausbeute (*recall*)

RAG Retrieval-Augmented Generation

TBD Trainings-/Beispieldaten

TL Nachverfolgbarkeitsverbindung (*trace link*)

TLR Wiederherstellung von Nachverfolgbarkeitsverbindungen (*traceability link recovery*)

TN Richtig negativ (*true negative*)

TP Richtig positiv (*true positive*)

1. Einleitung

Bei der Entwicklung und Wartung von Softwareprojekten entstehen verschiedene Artefakte, wie beispielsweise Anforderungen. Diese Anforderungen können in verschiedenen Beziehungen zueinanderstehen, da sie meistens nicht unabhängig voneinander sind [37]. Bei Beziehungen zwischen High-Level-Anforderungen (engl.: *high-level requirements*, HLRs) und Low-Level-Anforderungen (engl.: *low-level requirements*, LLRs) kann es sich beispielsweise um Verfeinerungen handeln [37]. Um diese Beziehungen zwischen Anforderungen zu repräsentieren, verwendet man in der Softwareentwicklung standardmäßig TLs [14]. Werden TLs zwischen HLRs und LLRs innerhalb eines Softwareprojektes vollständig erfasst und dokumentiert, bieten sie viele verschiedene Vorteile [30].

In der Realität werden TLs zwischen HLRs und LLRs jedoch häufig gar nicht oder nur inkonsistent erfasst, da der Aufwand bei der manuellen Ermittlung sehr groß ist [16, 25, 30]. Aus diesem Grund beschäftigt sich die Forschung mit der automatisierten Wiederherstellung von Nachverfolgbarkeitsverbindungen (engl.: *traceability link recovery*, TLR) von HLRs zu LLRs, für die es verschiedene Ansätze gibt [18, 30]. In neueren automatisierten Ansätzen für die TLR von HLRs zu LLRs kommen vermehrt vortrainierte Sprachmodelle (engl.: *pretrained language models*, PLMs) zum Einsatz, da sie in der Regel zu besseren Ergebnissen führen [21, 24]. Bei der Verwendung von PLMs für die TLR zwischen HLRs und LLRs kommt entweder Feinanpassung zum Einsatz, bei der das Modell vor der Verwendung angepasst wird, oder Prompting, bei dem das PLM ohne weitere Anpassung genutzt wird [21, 24]. Für Feinanpassung und *few-shot*/*multi-shot* Prompting werden zwangsläufig annotierte Daten beziehungsweise Trainings-/Beispieldaten (TBD) (TLs und Nicht-TLs) benötigt, für *zero-shot* Prompting hingegen nicht.

Bei der TLR treten in der Realität sehr unterschiedliche Situationen auf. Einerseits gibt es Projekte, in denen bislang keine TLs erfasst wurden. In diesen müssen also alle TLs ermittelt werden (TL-Generierung). Andererseits gibt es Projekte, bei denen bereits ein Teil der TLs vorliegt und vervollständigt werden muss (TL-Vervollständigung). Zusätzlich kann es vorkommen, dass vollständig annotierte Projekte zur Verfügung stehen, bei denen also bereits alle TLs ermittelt wurden. Aufgrund der unterschiedlichen Voraussetzungen der Ansatzarten können nicht alle Ansätze in allen realistischen Szenarien gleichermaßen verwendet werden. Außerdem unterscheiden sich die verschiedenen Feinanpassungs- und Prompting-Ansätze in ihrer Leistung. Teilweile zeigt sich dieser Leistungsunterschied auch, wenn andere Daten zum Training oder als Beispiele in Prompts genutzt werden. Dies wirft die zentrale Frage auf, welcher Ansatz unter welchen Bedingungen die besten Ergebnisse liefert. Diese Frage wurde noch nicht von der Forschungsgemeinschaft beantwortet, da

bislang kein umfassender Vergleich zwischen aktuellen Feinanpassungs- und Prompting-Ansätzen durchgeführt wurde. Deshalb kann ein Praktiker aktuell nicht abschätzen, welcher Ansatz die beste Leistung in seinem Anwendungsfall bringt.

Das Hauptziel dieser Bachelorarbeit ist es daher, zu ermitteln, mit welcher TBD-Anzahl welcher Ansatz in welchem Anwendungsszenario die beste Performance erzielt. Dazu wird ein systematischer Vergleich zwischen Feinanpassungs- und Prompting-Ansätzen für die TLR von HLRs zu LLRs durchgeführt. Zunächst werden geeignete Ansätze für die Untersuchungen ermittelt. Anschließend wird der eigentliche Vergleich in vier verschiedenen, realistischen Szenarien durchgeführt, die unterschiedliche Anwendungssituationen der TLR abbilden. In diesen werden TL-Generierung und TL-Vervollständigung mit und ohne die Nutzung anderer Projekte durchgeführt. In den Experimenten der Szenarien wird, wenn möglich, die Anzahl der verfügbaren TBD systematisch variiert und ihr Einfluss auf die Leistung der Ansätze evaluiert. Im Anschluss werden die Ergebnisse der verschiedenen Ansätze für jedes Szenario vergleichend ausgewertet.

Die Bachelorarbeit ist wie folgt aufgebaut: In Kapitel 2 werden wichtige Grundlagen beschrieben und in Kapitel 3 wird der aktuelle Forschungsstand vorgestellt. In Kapitel 4 wird eine begründete Auswahl der TLR-Ansätze aus der Forschung vorgenommen. Zusätzlich wird die Implementierung der ausgewählten Ansätze dargestellt. In Kapitel 5 werden die Szenarien und Experimente beschrieben und ausgewertet. In Kapitel 6 werden Bedrohungen der Validität und Limitierungen der Arbeit aufgezeigt. Außerdem wird ein Ausblick gegeben. Im abschließenden Kapitel 7 wird die Bachelorarbeit zusammengefasst und zusätzlich werden Schlussfolgerungen gezogen.

2. Grundlagen

In diesem Kapitel werden Grundlagen vorgestellt, welche für das Verständnis der Bachelorarbeit benötigt werden. Ziel ist es, relevante Begriffe zu definieren und wichtige Konzepte vorzustellen.

2.1. Nachverfolgbarkeit

Nachverfolgbarkeit (engl.: *traceability*) beschreibt die Möglichkeit, Artefakte aus der Softwareentwicklung (Quelltext, Anforderungen, Testfälle, usw.) miteinander in Beziehungen zu setzen und diese Beziehungen zu analysieren [14].

2.1.1. Nachverfolgbarkeitsverbindungen

Um diese Beziehungen zwischen Artefakten darzustellen, zu repräsentieren, zu dokumentieren und zu analysieren werden in der Softwareentwicklung TLs verwendet [14]. Ein TL verbindet immer jeweils genau zwei Artefakte miteinander [14].

TLs haben je nach Kontext und Artefakttypen unterschiedliche Bedeutungen [14, 21]. In dieser Bachelorarbeit sind ausschließlich HLRs und LLRs relevant. HLRs beschreiben das System auf einer allgemeinen, übergeordneten Ebene [32]. LLRs werden aus HLRs abgeleitet [32], sind deutlich spezifischer und beinhalten technische Details [32]. Sie werden von der Forschungsgemeinschaft teilweise auch Designartefakt genannt [12, 24, 40]. Bei TLs zwischen HLRs und LLRs handelt es sich demnach um Verfeinerungen bzw. um Vorgänger/Nachfolger-Beziehungen [37].

2.1.2. Wiederherstellung von Nachverfolgbarkeitsverbindungen

In der Praxis werden TLs gar nicht oder nur inkonsistent dokumentiert [16, 25]. Dies erschwert die Nutzung und erfordert Ansätze zur nachträglichen TLR.

2.1.2.1. Definition

Formal sieht diese Aufgabe wie folgt aus:

- Gegeben: Eine Menge Quellartefakte (QM) und eine Menge Zielartefakte (ZM) [14, 16].
- Gesucht: Eine vollständige Menge an TLs zwischen QM und ZM (TLM), wobei $TLM \subseteq QM \times ZM$ [14, 16].

In der Praxis werden bei einigen Ansätzen zusätzlich noch annotierte Daten benötigt, z.B. zum Trainieren von Modellen [24, 25]. Annotierte Daten für die TLR bestehen aus Paaren von Quell- und Zielartefakten sowie der Information, ob zwischen diesen eine TL existiert oder nicht.

2.1.2.2. Unterscheidungen

Die TLR kann man in verschiedene Arten unterteilen. Man kann TLs manuell, automatisiert und semi-automatisiert ermitteln [14]. Bei der manuellen TLR ermittelt ein Mensch die TLs [14], was sehr hohe Kosten verursachen kann [16, 30]. Bei der automatisierten TLR wird kein Mensch benötigt, da die TLR mit automatisierten Techniken bzw. Werkzeugen durchgeführt wird [14]. Die semi-automatisierte TLR ist eine Kombination der automatisierten und manuellen TLR [14]. Die TLR-Ansätze, welche in dieser Arbeit untersucht werden, sind automatisiert.

Es gibt zusätzlich eine Unterscheidung der TLR-Aufgabenart. Bei der TL-Generierung müssen alle TLs eines Projekts ermittelt werden [24]. Bei der TL-Vervollständigung hingegen liegt bereits eine unvollständige Menge an TLs aus dem Projekt vor und die restlichen TLs müssen ermittelt werden [24].

2.2. Vortrainierte Sprachmodelle

PLMs sind Modelle, die Sprachstrukturen und Wahrscheinlichkeitsverteilungen innerhalb von Texten erfassen [3, 41]. Es gibt eine große Bandbreite an verschiedenen PLMs. Im Rahmen dieser Arbeit bezieht sich dieser Begriff jedoch ausschließlich auf Modelle, die auf der Transformer-Architektur [41] basieren, wie z.B. BERT [8] oder GPT-3 [4].

Diese Modelle wurden bereits auf einer sehr großen Menge an Textdaten ohne manuelle Annotation vortrainiert, wodurch sie ein allgemeines Verständnis von natürlicher Sprache erlangten [4, 8, 25]. Das auf diese Weise erworbene Sprachverständnis kann vom Modell auf unterschiedliche Aufgaben, wie z.B. die TLR, übertragen werden [25, 40].

2.2.1. Modellarten

Aus der Transformer-Architektur [41] sind verschiedene Arten von PLMs entstanden. Die drei wichtigsten PLM-Arten verwenden unterschiedliche Teile der Architektur [26]. Encoder verwenden den Encoder-Teil vom Transformer [41], Decoder nutzen den Decoder-Teil und Encoder-Decoder implementieren die vollständige Architektur [26]. Letztere Modellart ist nicht weiter relevant für diese Bachelorarbeit, da sie hauptsächlich für Text-zu-Text Aufgaben, wie z.B. Übersetzung, verwendet wird [41].

2.2.1.1. Encoder

Encoder erstellen eine kontextuelle Repräsentation von Texten [8]. Sie werden für verschiedene Aufgaben wie Lückenfüllung in Texten, Textklassifikation und Vorhersage nächster Sätze genutzt [8, 36, 43].

Bei der Verwendung von Encodern wird der Text zuerst in Tokens (Wörter, Wörterpräfixe, Wörtersuffixe, Wortteile, Buchstaben, Satzzeichen, usw.) umgewandelt [41]. Diese Tokens besitzen Vektorrepräsentationen, welche an das Modell übergeben und mit Positionsinformationen kombiniert werden [41]. Nachdem das Modell durchlaufen wurde, gibt es für jeden Token einen Vektor aus, welcher die Bedeutung des Tokens im Kontext des Eingabetexts beschreibt [8].

Beim BERT-Modell (Encoder) [8], welches unter anderem in dieser Arbeit verwendet wird, gibt es spezielle Token, welche nicht aus dem Eingabetext extrahiert werden [8]. Diese werden beispielsweise für die Textklassifikation (CLS-Token), zum Beenden oder Abtrennen von Textsegmenten (SEP-Token) oder für das Auffüllen von zu kurzen Texten (PAD-Token) verwendet [8]. Zusammen kann man bei BERT [8] maximal 512 Tokens übergeben [8].

2.2.1.2. Decoder

Decoder sind autoregressive Modelle, welche für die Texterzeugung genutzt werden [4, 41]. Da verschiedene Aufgaben als Text formuliert werden können, kann man Decoder auch für die Lösung verschiedener anderer Aufgaben nutzen [4, 39].

Für die Nutzung von Decodern wird der Prompt (Eingabetext mit der Aufgabenbeschreibung) ebenfalls zuerst in Tokens und dann in Vektorrepräsentationen umgewandelt [4, 41]. Diese Repräsentationen werden auch wieder mit Positionsinformationen kombiniert und dann dem Modell als vorherige Ausgabe übergeben [4, 39, 41]. Das Modell ermittelt dann ein nächstes Token [4, 41]. Welches Token ausgewählt wird, hängt nicht nur von den vorherigen Tokens ab, sondern auch von anderen Faktoren, wie dem Zufallswert (engl.: *random seed*) [35]. Das ermittelte Token wird danach an den Prompt angefügt und dem Modell als neue vorherige Ausgabe übergeben [4, 41]. Dies wird so lange wiederholt, bis das Modell ein *End-Of-Text*-Token auswählt [4].

Die GPT-4o- und GPT-4o-mini-Modelle, welche in dieser Bachelorarbeit genutzt werden, ermöglichen die Verwendung von maximal 128.000 Tokens, wobei das Modell selbst die Ausgabe von maximal 16.384 Tokens erlaubt [34].

2.2.2. Verwendungsarten

Es gibt unterschiedliche Arten wie man PLMs nutzen kann [4, 8]. In dieser Bachelorarbeit wird zwischen Feinanpassung und Prompting unterschieden.

2.2.2.1. Feinanpassung

Bei der Feinanpassung (engl.: *fine-tuning*) wird ein PLM nicht direkt genutzt, sondern zuerst mit aufgaben- und/oder domänenspezifischen Daten an eine spezielle Zielaufgabe angepasst [8, 38]. Danach wird das angepasste Modell zur Lösung dieser Aufgabe verwendet [8, 25, 38].

Es gibt verschiedene Feinanpassungsansätze, welche unterschiedlich funktionieren [8, 22, 23, 27, 38]. Bei aktuell vorhandenen Ansätzen werden der Modellaufbau angepasst [22, 25], die Modellparameter aktualisiert [8, 25, 38] und/oder die Eingabedaten generisch verändert [23].

Um Feinanpassung durchführen zu können, wird ein Modell benötigt, welches auf einem PLM basiert [8, 24, 25]. In dieser Bachelorarbeit ist das Modell *BertForSequenceClassification* aus der Python-Bibliothek *transformers* relevant. Der Datenfluss dieses Modells ist in Abbildung 2.1 dargestellt. Es eignet sich für Feinanpassung von Textklassifikation [24, 25]. An das Modell übergibt man die Text-Tokens (Ermittelten Tokens eines Textes), welche mit einem CLS-Token am Anfang und einem SEP-Token und möglichen PAD-Tokens am Ende ergänzt werden [25]. Zuerst wird ein beliebiges BERT-Modell [8] durchlaufen [24, 25]. Anschließend wird die Ausgabe des CLS-Tokens des BERT-Modells [8] mit einer dichten neuronalen Schicht gebündelt (engl.: *pooling*) [25]. Die gebündelte Ausgabe wird danach mit einer Klassifikationsschicht auf einen n-dimensionalen Vektor reduziert und mit Anwendung der *Softmax*-Funktion in n-Klassenwahrscheinlichkeiten umgewandelt [25]. Die Klasse mit der höchsten Wahrscheinlichkeit ist das Klassifikationsergebnis [25].

Für die eigentliche Feinanpassung müssen neben der Modellwahl verschiedene andere Festlegungen getroffen werden [24, 25]. Eine benötigte Festlegung ist die Verlustfunktion [25]. Diese wird für die Berechnung und Minimierung des Trainingsfehlers benutzt. Bei *BertForSequenceClassification* kommt standardmäßig der *Cross-Entropy-Loss* als Verlustfunktion für binäre Klassifikation zum Einsatz:

$$L = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\right],$$

wobei N die Anzahl annotierter Trainingsdaten, $y \in \{0, 1\}$ das wahre Label und $\hat{y} \in (0, 1)$ die vorhergesagte Wahrscheinlichkeit darstellt [24, 25].

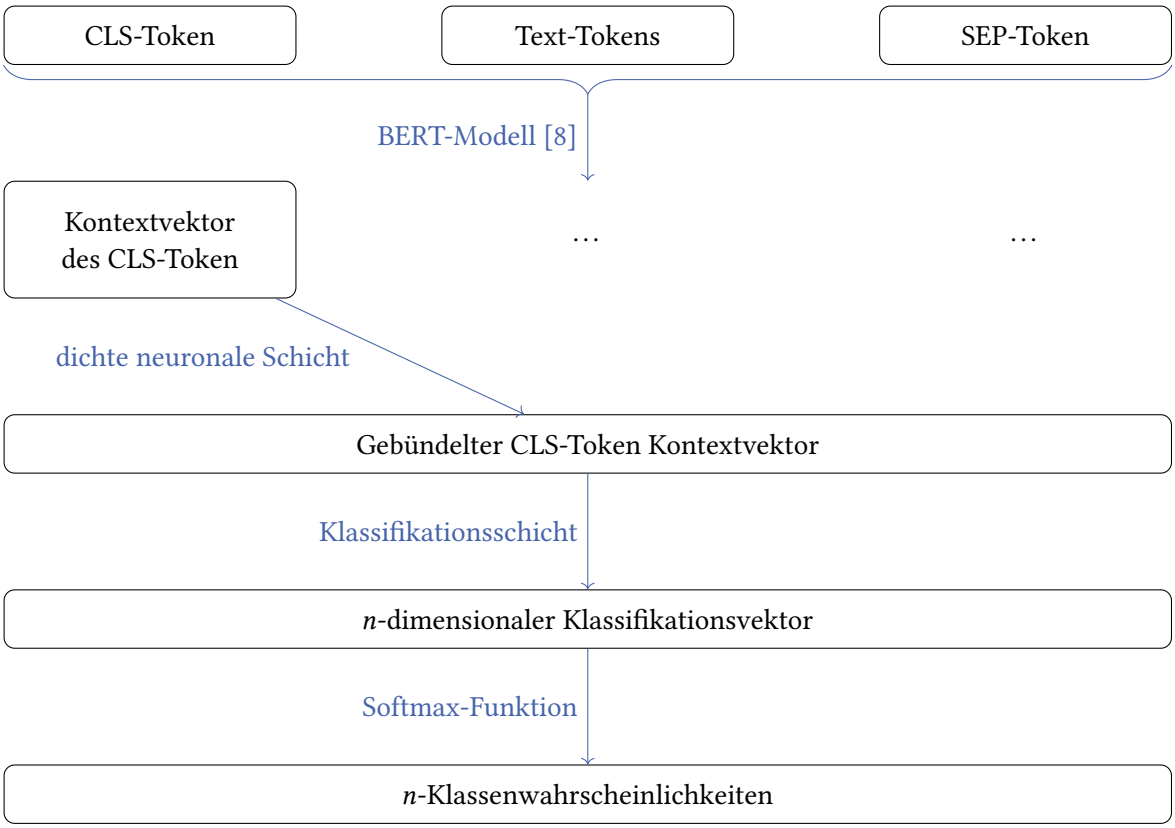

Abbildung 2.1.: Datenfluss des *BertForSequenceClassification*-Modells aus der Python-Bibliothek *transformers* - Modell für Textklassifikation - *n* entspricht der Klassenanzahl

Zusätzlich sollten die wichtigsten Hyperparameter festgelegt werden [6, 25]. Dies sind Parameter, welche den Trainingsprozess beeinflussen, aber nicht zu den eigentlichen Modellparametern gehören. Wichtige Hyperparameter sind folgende:

- Anzahl an Trainingsepochen (engl.: *number of training epochs*): Dieser Parameter gibt an, wie oft der Trainingsdatensatz während des Trainings durchlaufen wird. Wenn dieser Parameter zu niedrig gewählt wird, dann kann es zu Unteranpassung (engl.: *underfitting*) kommen, wobei das Modell die Daten nicht gut genug lernt. Wenn dieser Parameter zu hoch gewählt wird, dann kann es zu Überanpassung (engl.: *overfitting*) kommen, wobei sich das Modell zu stark an die Trainingsdaten anpasst und deswegen bei ungesehenen Daten verschätzt.
- Lernrate (engl.: *learning rate*): Dieser Parameter legt fest, wie groß die Veränderungsschritte der Modellparameter in jedem Trainingsschritt sind. Wenn dieser Parameter zu niedrig gewählt wird, dann kann das Training sehr lange dauern. Wenn dieser Parameter zu groß gewählt wird, dann kann die Modellkonfiguration mit dem minimalen Verlust in jedem Schritt übersprungen werden, wodurch der Verlust möglicherweise nicht reduziert wird.
- Gewichtsverfall (engl.: *weight decay*): Dieser Parameter schränkt die Modellkapazität ein, indem große Modellparameter zu einem höheren Verlust führen. Wenn dieser

Parameter zu niedrig gewählt wird, dann kann es zu Überanpassung kommen. Wenn dieser Parameter zu groß gewählt wird, dann kann es zu Unteranpassung kommen.

- Anzahl an Trainingsdaten pro Trainingsschritt pro Gerät (engl.: *per device train batch size*): Dieser Parameter gibt an, wie viele Trainingsdaten in jedem Trainingsschritt auf jedem Gerät (Prozessor oder Grafikkarte) genutzt werden. Wenn dieser Parameter zu niedrig gewählt wird, dann können die Modellparameterveränderungen rauschbehaftet sein, wodurch das Training länger dauern kann. Wenn dieser Parameter zu groß gewählt wird, dann kann das Training möglicherweise nicht ausgeführt werden, da der vorhandene Grafikspeicher eventuell überschritten wird.

Die Festlegung der Hyperparameter findet mittels der Hyperparameteroptimierung statt [6]. Für diese legt man eine Optimierungsstrategie fest, welche dazu genutzt wird, die Hyperparameter zu ermitteln, welche die besten Ergebnisse liefern [6]. Eine relevante Optimierungsstrategie ist *Grid Search*, wobei man eine Menge Hyperparameter und jeweils zugehörige Werte, welche überprüft werden sollen, nutzt [6]. Bei *Grid Search* werden die optimalen Hyperparameter ermittelt, indem alle möglichen Kombinationen aus den zugehörigen Werten getestet und die Hyperparameterkombination mit den besten Ergebnissen ausgewählt werden [6].

Zusätzlich kann eine Stichprobenstrategie (engl.: *sampling strategy*) verwendet werden [25]. In dieser Bachelorarbeit ist die Strategie *Dynamic Random Negative Sampling* relevant, da ein Ungleichgewicht zwischen den Klassen vorliegt und dieses mit dieser Strategie ausgeglichen werden kann [25]. Außerdem führt diese Strategie dazu, dass die Trainingsdauer pro Trainingsepoche stark reduziert wird, da beim *Dynamic Random Negative Sampling* in jeder Trainingsepoche nicht alle Daten benutzt werden, sondern alle Daten aus der kleineren Klasse und eine gleiche Anzahl zufällig ausgewählte Daten aus der größeren Klasse [25].

2.2.2.2. Prompting

Beim Prompting wird ein Decoder-PLM ohne weitere Anpassung des Modells verwendet, indem man die zu lösende Aufgabe in einen Prompt einbaut, an das PLM übergibt und einen Antworttext vom Modell erhält [11, 40].

Es existieren verschiedene Prompting-Techniken, welche teilweise miteinander kombiniert werden können [40]. Bei der *Chain-of-Thought*-Technik wird das Decoder-Modell beispielsweise dazu aufgefordert, seinen Denkprozess auszugeben, bevor es eine Antwort auf die Aufgabenstellung liefert [11, 21, 40]. Eine andere Technik heißt *few-shot* bzw. *multi-shot* Prompting [4]. Dabei enthält der Prompt zusätzlich zur Aufgabenstellung auch Beispiellösungen der Aufgabe [4]. Im Gegensatz dazu wird beim *zero-shot* Prompting nur die Aufgabenstellung übergeben [4].

2.3. Experimentelle Grundlagen

Für das Verständnis der Durchführung und Auswertung der Experimente dieser Bachelorarbeit wird zusätzliches Wissen über die genutzten Datenaufteilungsstrategien und verwendeten Evaluationsmetriken benötigt.

2.3.1. Datenaufteilungsstrategien

Einige Experimente in dieser Bachelorarbeit benötigen eine Aufteilung der Daten, da teilweise sowohl TBD als auch Testdaten benötigt werden. Diese Mengen müssen unterschiedliche Daten enthalten [6]. Grund dafür ist, dass die Ergebnisse sonst verfälscht werden, da ein PLM bei gesehenen Daten deutlich bessere Ergebnisse liefert als bei unbekannten Daten.

Es gibt verschiedene Strategien für die Datenaufteilung (DA) [6]. Einerseits kann jedes Projekt/jeder Datensatz einzeln betrachtet und aufgeteilt werden [6]. Hierbei muss zwischen stratifizierter und nicht-stratifizierter Aufteilung unterschieden werden [6]. Bei der stratifizierten DA sind die Klassenverhältnisse in allen Aufteilungsmengen gleich und bei der nicht-stratifizierten DA müssen diese nicht zwangsweise gleich sein [6]. Andererseits besteht die Möglichkeit, dass die Projekte entweder durchmischt oder einzeln als Aufteilungen betrachtet werden [6].

Darüber hinaus kann bei der DA eine Kreuzvalidierung durchgeführt werden [6]. Anstatt eine feste Test- und TBD-Menge zu verwenden, wird das Experiment mehrfach wiederholt, sodass jede Teilmenge der Daten einmal als Testmenge dient [6]. Bei Verwendung der Kreuzvalidierung wird die Performance ermittelt, indem der Durchschnitt von jeder einzelnen Evaluationsmetrik gebildet wird [6].

2.3.2. Evaluationsmetriken

Für die Auswertung und Bewertung der Experimente werden verschiedene Metriken verwendet. Da es sich bei der TLR um eine binäre Klassifikationsaufgabe handelt, können folgende Metriken direkt nach den Experimenten ermittelt werden:

- Richtig positiv (engl.: *true positive*, TP): Anzahl der Daten, welche vom TLR-Ansatz als TL klassifiziert wurden und in Wirklichkeit eine TL haben.
- Falsch positiv (engl.: *false positive*, FP): Anzahl der Daten, welche vom TLR-Ansatz als TL klassifiziert wurden und in Wirklichkeit keine TL haben.
- Richtig negativ (engl.: *true negative*, TN): Anzahl der Daten, welche vom TLR-Ansatz als Nicht-TL klassifiziert wurden und in Wirklichkeit keine TL haben.
- Falsch negativ (engl.: *false negative*, FN): Anzahl der Daten, welche vom TLR-Ansatz als Nicht-TL klassifiziert wurden und in Wirklichkeit eine TL haben.

Die zuvor angegebenen Metriken werden nicht für den direkten Vergleich zwischen TLR-Ansätzen genützt, da diese nur absolute und nicht relative Ergebnisse darstellen. Aus diesem Grund werden aus den oben angegebenen Metriken folgende Metriken abgeleitet, welche oft für die Bewertung von TLR-Ansätzen in der Forschung verwendet werden [11, 21, 24, 25]:

- Präzision (engl.: *precision*, P): $P = \frac{TP}{TP+FP}$

  Die Präzision gibt relativ an, wie viele als TL klassifizierte Daten in Wirklichkeit TL haben. Diese Metrik ist wichtig, da der Entwickler bei einer niedrigen Präzision das Vertrauen in die Klassifikation verliert und als TL klassifizierte Daten manuell überprüfen muss.

- Ausbeute (engl.: *recall*, R): $R = \frac{TP}{TP+FN}$

  Die Ausbeute gibt relativ an, wie viele der Daten, welche in Wirklichkeit eine TL haben, auch vom TLR-Ansatz erkannt wurden. Diese Metrik ist wichtig, da der Entwickler bei einer niedrigen Ausbeute möglicherweise wichtige Zusammenhänge übersieht, da eventuell wichtige TLs nicht vom TLR-Ansatz erkannt wurden.

- F_β-Metrik: $F_\beta = (1 + \beta^2) \cdot \frac{P \cdot R}{\beta^2 \cdot P + R}$

  Die F_β-Metrik kombiniert die Präzision und die Ausbeute. Mit β wird angegebenen, wie wichtig die Ausbeute im Vergleich zur Präzision ist. Wenn $\beta = 1$, ist die Ausbeute gleich gewichtet wie die Präzision und in diesem Fall handelt es sich um den harmonischen Durchschnitt der beiden Werte. Diese Metrik ist wichtig, da sowohl die Präzision als auch die Ausbeute wichtig für die TLR ist [21]. In dieser Arbeit sind sowohl die F_1- als auch die F_2-Werte relevant, da diese beiden F_β-Werte meistens für die Bewertung von TLR-Ansätzen in der Forschung verwendet werden [9, 11, 33].

3. Verwandte Arbeiten

In diesem Kapitel werden wissenschaftliche Veröffentlichungen vorgestellt, die im Zusammenhang mit dieser Bachelorarbeit stehen. Ziel ist es, den aktuellen Forschungsstand vorzustellen, meine Bachelorarbeit von ähnlichen Arbeiten abzugrenzen und dadurch die aktuell bestehende Forschungslücke aufzuzeigen.

3.1. Wiederherstellung von Nachverfolgbarkeitsverbindungen

Für die TLR gibt es verschiedenste Ansätze, welche mit der Zeit von der Forschung immer weiter entwickelt wurden.

3.1.1. Information Retrieval

Anfänglich lag der Schwerpunkt der TLR-Forschung auf Information Retrieval (IR), wobei versucht wird, TLs anhand textueller Ähnlichkeiten zu erkennen [16]. Dies zeigt sich exemplarisch an Veröffentlichungen wie Antoniol u. a. [2] und Hayes, Dekhtyar und Sundaram [17].

Mit der Zeit wurden diese Ansätze immer weiter verbessert und/oder erweitert. Zum Beispiel erstellten Moran u. a. [31] Comet. Dies ist ein TLR-Ansatz, welcher ein bayesianisches hierarchisches Modell in Kombination mit verschiedenen IR-Ansätzen verwendet. Es zeigte sich, dass die Leistung von Comet, verglichen mit einzelnen IR-Ansätzen, besser ist.

Ein anderer IR-Ansatz wurde von Hey u. a. [20] erstellt. Sie entwarfen den FTLR-Ansatz, welcher für die TLR von Anforderungen zu Quelltext entwickelt und getestet wurde. Bei diesem Ansatz teilten die Autoren die Artefakte in Teile (Sätze/Methoden) auf, ermittelten Vektoren mittels Worteinbettungen und bestimmten damit die Ähnlichkeit der Artefaktteile. Für die Bestimmung der Ähnlichkeit wurde als Funktion nicht die häufig verwendete Kosinus-Ähnlichkeit, sondern die *Word Mover's Distance* genutzt. Danach wurde für jede Methode der ähnlichste Satz aus jeder Anforderung ermittelt, mit einem unteren Grenzwert gefiltert und TL-Kandidaten pro Klasse durch Mehrheitsentscheid aggregiert. Diese TL-Kandidaten wurden anschließend mit einem unteren Ähnlichkeitsgrenzwert gefiltert. Sie zeigten, das FTLR bessere Ergebnisse erzielt, als vorhandene Ansätze, welche ebenfalls keine Trainingsdaten benötigen. Zusätzlich zeigte Hey [18], dass sowohl die genutzte Aufteilung und Aggregation als auch die verwendete Ähnlichkeitsfunktion die Leistung signifikant verbesserten. Später konnte der FTLR-Ansatz noch von Hey, Keim und Corallo

[19] verbessert werden, indem die Anforderungen unter anderem mit einem feinangepassten PLM vorgefiltert wurden.

Ein weiterer IR-Ansatz namens TRIAD wurde von Gao u. a. [12] ebenfalls für die TLR von Anforderungen zu Quelltext entworfen. Die Autoren verwendeten LLRs als Zwischenartefakte. Aus den Quell-, Zwischen- und Zielartefakten wurden Wortpaare extrahiert und in die Quell- und Zielartefakte eingesetzt. Zusätzlich nutzten sie transitive TLs (indirekte Verbindungen über Zwischenartefakte) für die Ermittlung von den TLs. Auch Gao u. a. [12] zeigten eine Verbesserung der Performance verglichen mit anderen IR-Ansätzen.

3.1.2. Klassisches maschinelles Lernen

Neben dem IR wurden auch maschinelles Lernen (engl.: *machine learning*, ML) für die TLR verwendet [16]. Im großen Unterschied zu den IR-Ansätzen benötigen diese annotierte Daten zum Trainieren. Guo, Cheng und Cleland-Huang [15] trainierten und verwendeten beispielsweise rekurrente neuronale Netze für die TLR. Dabei lieferte dieser Ansatz signifikant bessere Ergebnisse als vorhandene IR-Ansätze, welche zu der Zeit führend waren.

Ein anderer ML-Ansatz namens TRAIL wurde von Mills, Escobar-Avila und Haiduc [29] entworfen. Die Autoren ermittelten für jeden möglichen TL unterschiedliche Merkmale, womit verschiedene Modelle trainiert wurden. Dabei zeigte sich, dass das *Random Forest*-Modell für diesen Ansatz am besten geeignet ist und das verschiedene IR-Ansätze in der Leistung deutlich übertroffen werden konnten. Später verbesserten Mills u. a. [30] diesen Ansatz, indem sie durch aktives Lernen (engl.: *active learning*) die Anzahl der benötigten Trainingsdaten verringerten.

In dieser Bachelorarbeit wird kein IR- und ML-Ansatz in den Vergleich mit eingebunden, da sich zeigte, dass PLM-basierte Ansätze meist bessere Ergebnisse liefern [1, 11].

3.1.3. Feinanpassung

Eine Gruppe von PLM-basierten Ansätzen, welche annotierte Daten zum Trainieren benötigen, verwenden Feinanpassung.

Lin u. a. [25] veröffentlichten eine Publikation dazu. Sie entwarfen und untersuchten das T-BERT-Framework mit drei Feinanpassungsansätzen, wobei die Quellartefakte Fehlerberichte und Feature-Anfragen (engl.: *issues*, Issues) und die Zielartefakte Quelltext-Änderungen (engl.: *commits*, Commits) waren. Dabei kam heraus, dass die evaluierte Single-BERT-Variante die besten Ergebnisse liefert und die Siamese-BERT-Variante das beste Leistung/Kosten-Verhältnis hat.

Feinanpassungsansätze wurden ebenfalls von Lin u. a. [24] für ihre wissenschaftliche Veröffentlichung untersucht. Es wurden fünf Ansätze für die TLR von HLRs zu LLRs getestet, welche unterschiedliche Transferlernansätze für ein Zwischentraining nutzen. Lin u. a. [24] fanden dabei heraus, dass das Task-CLS-Modell, welches auf dem Single-BERT-Modell

von Lin u. a. [25] aufbaut, die besten Ergebnisse liefert. Außerdem ermittelten sie, dass das BERT-Modell [8] als Grundlage bei der Vervollständigung von TLs am besten funktioniert.

Auch Majidzadeh, Ashtiani und Zakeri-Nasrabadi [28] verwendeten Feinanpassung für die TLR. Sie kombinierten Feinanpassung eines Modells mit verschiedenen Datenaugmentierungstechniken und testeten es auf der TLR von Dokumentationen zu Methoden, von Issues zu Commits und von Issues zu Methoden. Sie fanden heraus, dass ihr Ansatz besser ist als verfügbare Ansätze, wie z.B. auch T-BERT von Lin u. a. [25].

Sowohl Deng u. a. [7] als auch Wang u. a. [43] verwendeten einen anderen Ansatz für die TLR von Issues zu Commits. Die Autoren der beiden Publikationen lösten die TLR mit einer Lückentextaufgabe und bauten die Textlücke in verschiedene Prompts ein. Dazu testeten sie eine Trainingsstrategie, bei welcher leichtes Rauschen in die Eingabe mit einfließt. Es zeigte sich, dass die Trainingsstrategie zu besseren Ergebnissen führt. Zugleich ermittelten sie, dass die Mittelung der Ergebnisse der einzelnen Prompts die beste Leistung erbringt. Beim Vergleich mit anderen Ansätzen zeigte dieser Ansatz signifikant bessere Ergebnisse.

Nur eine Publikation untersuchte bis jetzt die TLR zwischen Anforderungen. Dazu verglichen alle zuvor genannten Veröffentlichungen ihre Ansätze auch nicht mit Prompting.

3.1.4. Prompting

Prompting ist ein anderer Ansatz, welcher auf PLMs basiert. Auch Prompting wurde wie Feinanpassung schon mehrfach von der Forschungsgemeinschaft für die TLR verwendet und untersucht.

Rodriguez, Dearstyne und Cleland-Huang [40] testeten verschiedene Prompting-Techniken für die TLR von HLRs zu LLRs, von Anforderungen zu Klassen und von LLRs zu Klassen systematisch. Betrachtet wurden unterschiedliche sortierungs- und klassifikations-basierte Ansätze. Dabei stellte sich heraus, dass klassifikations-basierte Ansätze besser funktionieren als sortierungs-basierte Ansätze. Bei den klassifikations-basierten Ansätzen zeigte sich, dass die *Chain-of-Thought*-Technik leicht bessere Ergebnisse liefert als die anderen untersuchten Techniken.

Fuchß u. a. [11] untersuchten ebenfalls Prompting und erstellten das LiSSA-Framework. Sie erweiterten Prompting mit Retrieval-Augmented Generation (RAG) und überprüften diesen Ansatz auf der TLR von Anforderungen zu Quelltext, von Dokumentationen zu Quelltext und von Dokumentationen zu Modellen. Geprüft wurden nur klassifikations-basierte Prompting-Ansätze, welche aus den Ergebnissen von Rodriguez, Dearstyne und Cleland-Huang [40] abgeleitet wurden. Bei der Untersuchung wurde ersichtlich, dass hier auch die *Chain-of-Thought*-Technik bessere Ergebnisse liefert und dass ihr Ansatz im Durchschnitt besser ist als andere Ansätze, welche zu der Zeit führend waren. Hey u. a. [21] arbeiteten mit den Ergebnissen von Fuchß u. a. [11] weiter. Der Ansatz von Fuchß u. a. [11] wurde in dieser Veröffentlichung für die Wiederherstellung von TLs zwischen HLRs und LLRs getestet. Auch hier zeigte sich, dass die *Chain-of-Thought*-Technik zu besseren Ergebnissen führt und dass der Ansatz besser ist als andere Ansätze, welche zu der Zeit führend waren.

Niu u. a. [33] nutzten einen anderen RAG-Ansatz in Kombination mit Prompting wie Fuchß u. a. [11] und Hey u. a. [21]. Bei dieser Untersuchung wurde die Wiederherstellung und Validierung von TLs zwischen HLRs und LLRs von Kraftfahrzeugsystemen betrachtet. Der Fokus lag eher auf der Validierung als auf der Wiederherstellung, weswegen keine große Auswertung von der TLR-Performance stattfand.

Auch Fuchß u. a. [10] verwendeten den Ansatz von Fuchß u. a. [11] und Hey u. a. [21] als Grundlage und ersetzten den bisherigen Retrieval-Schritt durch ein mehrstufiges Filterverfahren, bei dem Prompting auf kleinen Decoder-PLMs durchgeführt wird. Sie testeten ihren Ansatz für die TLR von HLRs zu LLRs. Sie zeigten, dass die verkettete Filterung am besten funktioniert. Mit dieser Filterart konnten sie aktuelle IR-Ansätze in der Performance schlagen, aber nicht den ursprünglichen Ansatz von Fuchß u. a. [11] und Hey u. a. [21].

In dieser Bachelorarbeit führe ich im Unterschied zu Fuchß u. a. [10, 11], Hey u. a. [21], Niu u. a. [33] und Rodriguez, Dearstyne und Cleland-Huang [40] einen Vergleich mit einem Feinanpassungsansatz durch.

3.1.5. Vergleich von Feinanpassung und Prompting

Im Kontrast zu den vorherigen Publikationen evaluierten und verglichen zwei Veröffentlichungen sowohl Feinanpassung als auch Prompting für die TLR.

Bei der Arbeit von Etezadi u. a. [9] wurde Kashif und RICE für die TLR von Anforderungen zu gesetzlichen Vorschriften untersucht. Kashif ist ein Feinanpassungsansatz, welcher ein modifiziertes BERT Modell [8] verwendet, und RICE ist ein *few-shot* Prompting-Ansatz. Die Auswertung zeigte, dass RICE generell deutlich bessere Ergebnisse liefert als Kashif.

Auch Ge u. a. [13] verglichen die beiden Ansatzarten miteinander, wobei aber der Hauptfokus auf der Feinanpassung lag. Sie führten die TLR von HLRs zu LLRs auf zehn Datensätzen durch, wobei vier Datensätze unterschiedlich aus den anderen sechs Datensätzen zusammengesetzt wurden. Ge u. a. [13] variierten mit dem eingesetzten Modell, der verwendeten Feinanpassungsstrategie und dem genutzten Prompt. Außerdem verwendeten sie verschiedene Zusammenfassungs- und Datenaugmentierungstechniken. Beim Vergleich zwischen Prompting und Feinanpassung variierten die Autoren aber nicht mit der Anzahl annotierter Daten innerhalb der Datensätze, sondern verwendeten stattdessen eine feste Datenaufteilung. Bei ihnen zeigte sich, dass die Verwendung von Zusammenfassungs- und Datenaugmentierungstechniken zu besseren Ergebnissen führt. Dazu ermittelten sie, dass der Feinanpassungsansatz Prompt-Tuning [23] in Kombination mit dem genutzten 7B-LLaMA-Modell der beste Ansatz ist. Anders als bei Etezadi u. a. [9] war bei Ge u. a. [13] der Feinanpassungsansatz also besser als Prompting.

In Abgrenzung zu der Arbeit von Etezadi u. a. [9] werden in dieser Bachelorarbeit die TLR von HLRs zu LLRs und *zero-shot* Prompting betrachtet. Im Gegensatz zu Ge u. a. [13] wird auch *few-shot*/*multi-shot* Prompting mit in den Vergleich einbezogen. Des Weiteren grenzt sich diese Bachelorarbeit von beiden Arbeiten ab, indem systematisch Prompting und Feinanpassung mit unterschiedlicher Anzahl annotierter Daten verglichen wird.

3.2. Vergleich von Feinanpassung und Prompting

Feinanpassung und Prompting wurden nicht nur für die TLR untersucht, sondern auch für eine Vielzahl anderer Aufgaben.

So verglichen beispielsweise Chen, Yi und Varró [5] beide Ansätze bei der Erstellung von Taxonomien auf zwei Datensätzen unterschiedlicher Größe. Ein Datensatz ist domänenunabhängig und umfasst Taxonomien mit „ist-ein"-Beziehungen (WordNet) und der andere Datensatz bildet hierarchische Beziehungen zwischen Informatik-Konzepten ab (ACM CCS). Prompting lieferte bei dem kleineren Datensatz (WordNet) deutlich bessere Ergebnisse, weil dort weniger Trainingsdaten verfügbar waren. Bei dem größeren Datensatz (ACM CCS) wurden hingegen gleich gute Ergebnisse wie Feinanpassung erzielt.

Auch bei Pecher, Srba und Bielikova [36] wurde ein ähnlicher Zusammenhang sichtbar, diesmal bei verschiedenen Textklassifikationsaufgaben auf acht Datensätzen, die sowohl binäre Klassifikationen (z.B. Erkennung grammatikalischer Korrektheit von Sätzen) als auch Mehrklassenklassifikationen (z.B. Klassifikation von Fragearten) umfassten. Ihr Vergleich ergab, dass Feinanpassung ab durchschnittlich 30 annotierten Daten bessere Ergebnisse erzielt als *zero-shot* Prompting und ab durchschnittlich 100 annotierten Daten auch besser wird als *few-shot* Prompting. Dazu stellten Pecher, Srba und Bielikova [36] fest, dass die Ergebnisse stark vom verwendeten Datensatz abhängen.

Ein weiterer Vergleich wurde von Walsh u. a. [42] durchführt und veröffentlicht. Sie untersuchten Feinanpassung und *few-shot* Prompting im Kontext der Bewertung von Kurzantworten. Bei ihnen zeigte sich eine deutliche Verbesserung von Feinanpassung gegenüber *few-shot* Prompting schon ab ca. 150 Trainingsdaten beim GPT-4o-mini-Modell.

Insgesamt verdeutlichen die Ergebnisse, dass Feinanpassung mit wachsender Anzahl annotierter Daten besser wird und ab einer gewissen Anzahl auch besser wird als Prompting.

3.3. Zusammenfassung

Die wichtigsten verwandten Arbeiten sind zusammengefasst in Tabelle 3.1 dargestellt. Hierbei handelt es sich um Veröffentlichungen, welche Feinanpassung und/oder Prompting für die TLR nutzten. Es gibt jeweils fünf Veröffentlichungen, welche nur Feinanpassung oder nur Prompting untersuchten. Nur zwei Publikationen untersuchten sowohl Feinanpassung als auch Prompting. Feinanpassung wurde bei zwei verwandten Arbeiten für die TLR von HLRs zu LLRs, welche in dieser Arbeit untersucht wird, evaluiert. Prompting evaluierten fünf verwandte Publikationen für die TLR von HLRs zu LLRs.

Veröffentlichung	TL-Typen	Feinanpassung	Prompting
Lin u. a. [25]	Issue → Commit	✓	
Lin u. a. [24]	HLR → LLR	✓	
Majidzadeh, Ashtiani und Zakeri-Nasrabadi [28]	Dokumentation → Methode Issue → Commit Issue → Methode	✓	
Deng u. a. [7]	Issue → Commit	✓	
Wang u. a. [43]	Issue → Commit	✓	
Rodriguez, Dearstyne und Cleland-Huang [40]	HLR → LLR Anforderung → Klasse LLR → Klasse		✓
Fuchß u. a. [11]	Anforderung → Quelltext Dokumentation → Quelltext Dokumentation→ Modell		✓
Hey u. a. [21]	HLR → LLR		✓
Niu u. a. [33]	HLR → LLR		✓
Fuchß u. a. [10]	HLR → LLR		✓
Etezadi u. a. [9]	Anforderung → gesetzliche Vorschrift	✓	✓
Ge u. a. [13]	HLR → LLR	✓	✓

Tabelle 3.1.: Übersicht der wichtigsten verwandten Arbeiten

4. Analyse und Implementierung der TLR-Ansätze

Das zentrale Ziel dieser Bachelorarbeit besteht darin, zu ermitteln, unter welchen Bedingungen man eher Feinanpassung oder eher Prompting für die TLR von HLRs zu LLRs verwenden sollte. Um dieses Ziel erreichen zu können, werden für den Vergleich zuerst Prompting- und Feinanpassungs-TLR-Ansätze benötigt. Von den in Kapitel 3 vorgestellten TLR-Ansätzen aus der Forschung für Feinanpassung und Prompting können aber nicht alle mit in den Vergleich einbezogen werden, da dies einerseits den Rahmen dieser Arbeit überschreiten würde und da anderseits nicht alle gefundenen Ansätze für die TLR von HLRs zu LLRs eingesetzt werden können.

Aus diesem Grund wird in diesem Kapitel eine Analyse der vorhandenen Ansätze und eine begründete Auswahl der TLR-Ansätze vorgenommen. Dafür werden zunächst die Kriterien, welche für die Ansatzwahl genutzt werden, aufgezeigt. Danach wird für jeweils Feinanpassung und Prompting begründet, welche entwickelten und untersuchten TLR-Ansätze mit in den Vergleich einbezogen werden. Zusätzlich wird in diesem Kapitel die Implementierung der gewählten Ansätze dargestellt, da einige begründete Veränderungen bei der Implementierung vorgenommen werden müssen.

4.1. Analyse

Bei der Analyse und Auswahl der TLR-Ansätze liegt der Schwerpunkt darauf, dass sie für die TLR von HLRs zu LLRs geeignet sind. Dafür wird überprüft, ob die Ansätze bereits für die TLR von HLRs zu LLRs evaluiert wurden. Wenn dies nicht der Fall ist, dann werden diese Ansätze für den Vergleich nicht weiter in Betracht gezogen. Der Grund hierfür liegt darin, dass der Fokus dieser Bachelorarbeit auf der Evaluation bestehender Ansätze liegt und nicht auf der Anpassung von TLR-Ansätzen auf andere Artefakttypen. Wenn nach dieser Auswahl noch mehrere Ansätze übrig bleiben, dann wird die weitere Wahl anhand von anderen Faktoren getroffen, wie beispielsweise der Performance des Ansatzes oder wie stark der Fokus in der Veröffentlichung auf die TLR gelegt wurde.

4.1.1. Feinanpassung

Bei der Feinanpassung stehen die Ansätze von Deng u. a. [7], Etezadi u. a. [9], Ge u. a. [13], Lin u. a. [24, 25], Majidzadeh, Ashtiani und Zakeri-Nasrabadi [28] und Wang u. a. [43] zur Verfügung.

Von diesen wurden als einziges die Ansätze von Ge u. a. [13] und Lin u. a. [24] für die TLR von HLRs zu LLRs evaluiert. Dabei werden die Ansätze von Ge u. a. [13] in dieser Arbeit nicht weiter berücksichtigt, da sie leichtgewichtige Feinanpassungsmethoden wie LoRa [22], Prompt-Tuning [23] oder P-Tuning-v2 [27] verwenden und die Nutzung von mehreren Feinanpassungsansätzen den Rahmen der Arbeit überschreiten würde.

Die fünf verschiedenen TLR-Ansätze von Lin u. a. [24] nutzen alle Feinanpassung, bei der alle Modellparameter aktualisiert werden. Da die Evaluation von Lin u. a. [24] sowohl auf dieselbe Art und Weise als auch auf den gleichen Datensätzen/Projekten für alle Ansätze durchgeführt wurde, wird der Ansatz anhand der Performance ausgewählt. Der Task-CLS-Ansatz brachte die beste Performance, weswegen dieser in abgewandelter Form als Feinanpassungsansatz (FA) verwendet wird.

4.1.2. Prompting

Prompting-Ansätze wurden von Etezadi u. a. [9], Fuchß u. a. [10, 11], Ge u. a. [13], Hey u. a. [21], Niu u. a. [33] und Rodriguez, Dearstyne und Cleland-Huang [40] entwickelt und/oder evaluiert. Im Gegensatz zur Feinanpassung gibt es beim Prompting mehrere Ansätze, die dieselbe TLR-Aufgabe evaluierten, die in dieser Bachelorarbeit untersucht wird. Diese Evaluationen wurden von Fuchß u. a. [10], Ge u. a. [13], Hey u. a. [21], Niu u. a. [33] und Rodriguez, Dearstyne und Cleland-Huang [40] durchgeführt.

Eine weitere Ansatzwahl nach der Performance ist bei den meisten dieser Veröffentlichungen nicht sinnvoll, da häufig verschiedene Datensätze genutzt wurden und/oder nicht die gleichen Evaluationsmetriken vorliegen, weswegen die Ansätze hauptsächlich nach anderen Faktoren herausgefiltert werden. Die Ansätze von Rodriguez, Dearstyne und Cleland-Huang [40] werden nicht mit in den Vergleich einbezogen, da bereits neuere Ansätze von beispielsweise Fuchß u. a. [10], Ge u. a. [13] und Hey u. a. [21] entwickelt und evaluiert wurden, welche auf den Ergebnissen von Rodriguez, Dearstyne und Cleland-Huang [40] aufbauten. Auch der Ansatz von Niu u. a. [33] wird ebenfalls in dieser Arbeit nicht genutzt, da einerseits der Fokus nicht auf der TLR lag und andererseits dieser Ansatz nur für eine spezielle Domäne (Kraftfahrzeugsysteme) evaluiert wurde. Die Prompting-Ansätze von Ge u. a. [13] werden auch nicht mit einbezogen, da sie keine Analyse zur Ermittlung des besten Prompting-Ansatzes durchführten. Dadurch ist nicht bekannt, welcher der beste Prompting-Ansatz aus der Veröffentlichung ist. Diese Ermittlung kann auch nicht in dieser Bachelorarbeit durchgeführt werden, da die genauen Prompting-Ergebnisse nicht veröffentlicht wurden. Zusätzlich ist eine Betrachtung aller Prompting-Ansätze von Ge u. a. [13] ist nicht möglich, da dies den Rahmen dieser Arbeit überschreiten würde. Fuchß u. a. [10] und Hey u. a. [21] testeten auf den gleichen Datensätzen und gaben dieselben Evaluationsmetriken an,

wodurch die Ansatzwahl bei diesen Veröffentlichungen nach der Performance getroffen werden kann. Der Ansatz von Hey u. a. [21] brachte die bessere Leistung, weswegen als Prompting-Ansatz 1 (PA1) der LiSSA-Ansatz verwendet wird, welcher von Hey u. a. [21] für die TLR von HLRs zu LLRs getestet wurde.

Darüber hinaus wird der Vergleich um einen weiteren Prompting-Ansatz (Prompting-Ansatz 2 (PA2)) ergänzt, der ausschließlich auf Prompting basiert und keine zusätzlichen Verfahren wie RAG oder Datenaugmentierung einsetzt. Dadurch werden die Ergebnisse ausschließlich durch die jeweilige Methode bestimmt, was einen direkten Vergleich der Leistungsfähigkeit von Feinanpassung und Prompting ermöglicht. Dafür wird eine Abwandlung von dem PA1 verwendet.

In dieser Bachelorarbeit wird zusätzlich *few-shot*/*multi-shot* Prompting betrachtet. Der Grund dafür ist, dass diese Art von Prompting noch nicht von der Forschungsgemeinschaft für die TLR von HLRs zu LLRs evaluiert wurde und es möglicherweise eine Alternative zu Feinanpassung ist, da beide Ansatzarten annotierte Daten benötigen. Außerdem zeigten Etezadi u. a. [9] in der Vergangenheit, dass *few-shot* Prompting im Vergleich zur Feinanpassung bessere Ergebnisse liefern kann. Der Ansatz von Etezadi u. a. [9] ist auch der einzige *few-shot*/*multi-shot* Prompting-TLR-Ansatz aus der Forschung. Dieser fällt aber aus der Betrachtung raus, da er nicht für die TLR von HLRs zu LLRs entwickelt wurde. Deswegen wird als Prompting-Ansatz 3 (PA3) eine Abwandlung von PA2 genutzt.

4.2. Implementierung

Für den Vergleich werden die Ansätze, welche in Abschnitt 4.1 ausgewählt werden, teilweise in einer eigenen Implementierung umgesetzt. Dies ist notwendig, da Teile der Ansätze aus der Forschung in dieser Bachelorarbeit aus verschiedenen Gründen abgewandelt werden.

4.2.1. Feinanpassungsansatz: *BertForSequenceClassification*

Dieser Ansatz verwendet das Modell *BertForSequenceClassification*, welches in Unterunterabschnitt 2.2.2.1 bereits vorgestellt wurde. Das Modell wird bei diesem Ansatz mit zwei Klassen ($n = 2$) verwendet: Klasse 0 ist die Klasse der Nicht-TLs und Klasse 1 ist die Klasse der TLs. Als Eingabetext-Tokens in dieses Modell wird die zu überprüfende Kombination aus HLR und LLR übergeben. Dazu werden die beiden Artefakte zunächst einzeln in Tokens umgewandelt und dann durch ein SEP-Token getrennt. Nachdem das Modell durchlaufen wurde, wird die Klasse als Klassifikationsergebnis ausgewählt, die die größte Klassenwahrscheinlichkeit hat.

Beim Modell kann man verschiedene Kodierer-PLMs verwenden. Da sich zeigte, dass BERT bessere Ergebnisse liefert als andere Kodierer wie z.B. RoBERTa [24], wird bei diesem Ansatz BERT [8] verwendet. Als spezielle BERT-Variante wird *bert-large-cased* genutzt. *Large* wird verwendet, da im Vergleich zu *base* die Modellkapazität größer ist. *Cased* wird genutzt, da das Modell bei dieser Variante zwischen großen und kleinen Buchstaben unterscheidet.

Dies ist vermutlich wichtig, da in den vorhandenen Anforderungen der Datensätze viele großgeschriebene Abkürzungen vorkommen.

Beim originalen Ansatz von Lin u. a. [24] wurde vor der Feinanpassung auf dem verwendeten Modell ein Zwischentraining durchgeführt. Da die zwischentrainierten Modelle jedoch nicht veröffentlicht wurden und eine selbstständige Durchführung des Zwischentrainings den Rahmen der Bachelorarbeit überschreiten würde, wird auf diesen Schritt verzichtet.

Bei der eigentlichen Feinanpassung werden alle Modellparameter aktualisiert. Als Verlustfunktion kommt der für das Modell übliche *Cross-Entropy-Loss* zum Einsatz. Zusätzlich nutzt der originale Ansatz die Stichprobenstrategie *Online Negative Sampling*. Bei dieser Strategie sucht man in jedem Trainingsschritt die Nicht-TLs, die am wahrscheinlichsten als TL klassifiziert werden. Dies erhöht aber die Trainingszeit stark, da während des Trainings diese Nicht-TLs ermittelt werden müssen. Da in dieser Bachelorarbeit sehr viele verschiedene Modelle trainiert werden, ist eine Verwendung dieser Strategie aus diesem Grund nicht sinnvoll. Um die Vorteile einer Stichprobenstrategie nicht zu verlieren, wird in dieser Arbeit die Strategie *Dynamic Random Negative Sampling* verwendet, welche eine Alternative zum *Online Negative Sampling* ist [24].

4.2.2. Prompting-Ansatz 1: *zero-shot* mit Retrieval-Augmented Generation

Der LiSSA-Ansatz [11] muss für diese Bachelorarbeit nicht verändert werden. Bei diesem Ansatz werden zuerst mögliche Kombinationen aus Quell- und Zielartefakten herausgefiltert, indem die Artefakte in Vektoren mit dem *text-embedding-3-large* Einbettungsmodell umgewandelt und mit der Kosinus-Ähnlichkeit ($\frac{QA \cdot ZA}{\|QA\| \, \|ZA\|}$, wobei QA der Vektor des Quellartefakt und ZA der Vektor des Zielartefakt ist) die vier ähnlichsten Zielartefakte zu jedem Quellartefakt ermittelt werden. Nur diese ermittelten Kombinationen werden als mögliche TL-Kandidaten behandelt und dem Prompting unterzogen. Dabei können verschiedene Prompts und Modelle benutzt werden. Der *Chain-of-Thougth*-Prompt (Prompt 1) in Kombination mit dem GPT-4o-Modell (gpt-4o-2024-08-06) lieferte die besten Ergebnisse, weswegen nur diese Ansatzvariante in den Vergleich mit einbezogen wird.

Prompt 1: *zero-shot* Prompt
Below are two artifacts from the same software system. Is there a traceability link between (1) and (2)? Give your reasoning and then answer with 'yes' or 'no' enclosed in <trace> </trace>.

(1) requirement: "'[HLR]"'

(2) requirement: "'[LLR]"'

4.2.3. Prompting-Ansatz 2: *zero-shot* ohne Retrieval-Augmented Generation

Dieser Ansatz ist eine Abwandlung vom PA1. Bei PA2 wird die Vorfilterung der Kombinationen aus PA1 weggelassen, sodass alle mögliche Kombinationen aus Quell- und Zielartefakten aus einem Projekt dem Prompting unterzogen werden. Bei diesem Ansatz wird ebenfalls Prompt 1 und als Modell GPT-4o (gpt-4o-2024-08-06) genutzt.

4.2.4. Prompting-Ansatz 3: *few-shot*/*multi-shot*

Dieser Ansatz ist eine Abwandlung von PA2. Im Gegensatz zu PA2 nutzt dieser Ansatz einen *few-shot*/*multi-shot* Prompt (Prompt 2), welcher aus Prompt 1 abgeleitet wird. Als Modell wird GPT-4o-mini (gpt-4o-mini-2024-07-18) verwendet, da GPT-4o zu große Kosten bei Experimenten verursachen würde.

Prompt 2: *few-shot*/*multi-shot* Prompt
Below are examples of traceability link decisions between high-level and low-level requirements from the same software system.

Example 1:

(1) requirement: '"[HLR Beispiel 1]"'

(2) requirement: '"[LLR Beispiel 1]"'

Traceability link: [yes/no]

[Mögliche weitere Beispiele]

Now consider the following case. Is there a traceability link between (1) and (2)? Give your reasoning and then answer with 'yes' or 'no' enclosed in <trace> </trace>.

(1) requirement: '"[HLR]"'

(2) requirement: '"[LLR]"'

5. Szenarienbasierte Experimente und Auswertung

Für einen direkten Vergleich von Feinanpassung und Prompting benötigt man neben geeigneten Ansätzen zusätzlich auch eine Untersuchung ihrer Leistungsfähigkeit bei einer bestimmten Aufgabe. Beim FA, PA2 und PA3 liegen noch keine Ergebnisse vor, da diese Ansätze eigene Ableitungen/Implementierungen vorhandener Ansätze sind. Aus diesem Grund werden für die Ermittlung der Performance selbst Experimente durchgeführt, welche auf den gleichen Datensätzen/Projekten ausführt und mit identischen Evaluationsmetriken bewertet werden. Die Experimente orientieren sich an verschiedenen Szenarien. Diese bilden unterschiedliche reale Anwendungssituationen ab, in denen annotierte Projekte und projekt-interne Daten entweder vorhanden sind oder fehlen. Dadurch kann ein Entwickler in seinem Anwendungsfall mithilfe dieser Bachelorarbeit den TLR-Ansatz mit der besten Leistungsfähigkeit auswählen.

In diesem Kapitel werden zunächst die experimentellen Rahmenbedingungen beschrieben. Danach werden die einzelnen Szenarien vorgestellt. Wenn für die Szenarien neue Experimente durchgeführt werden, dann werden der Aufbau und die Ergebnisse dieser Experimente zusätzlich einzeln beschrieben. Anschließend werden in den Szenarien die Ergebnisse der verschiedenen Ansätze vergleichend ausgewertet. Bei der Auswertung wird als wichtigste Evaluationsmetrik der F_1-Wert verwendet, da diese Metrik für die vollautomatisierte TLR am wichtigsten ist [21]. Dazu werden die F_2-Werte bei der Auswertung mit beschrieben, da diese für die semi-automatisierte TLR eine hohe Relevanz besitzen [10, 21].

5.1. Experimentelle Rahmenbedingungen

Vor den Experimenten müssen verschiedene Rahmenbedingungen festgelegt werden. Hierzu zählen unter anderem die Datensätze, welche zum Trainieren und Testen der Ansätze verwendet werden, und die Hyperparameter für die Feinanpassung. Alle Experimente, welche durchgeführt werden, laufen auf einem Server mit einer NVIDIA Tesla V100S Grafikkarte mit 32 GB Grafikkartenspeicher. Für die DA, das eigentliche Training, das Testen der Modelle und die Darstellung der Ergebnisse wird die Programmiersprache Python genutzt. Beim Prompting wird die Temperatur auf null gesetzt. Zusätzlich wird ein fester Zufallswert verwendet, wodurch die Experimente möglichst deterministisch und reproduzierbar ablaufen.

Datensatz	Quellartefakte: HLRs	Zielartefakte: LLRs	Kombinationen	TLs
CM1-NASA	22	53	1166	45
Dronology	99	211	20889	220
GANNT	17	69	1173	68
Modis	19	49	931	41
WARC	63	89	5607	136

Tabelle 5.1.: Anzahl der Artefakte, Kombinationen und TLs in den Datensätzen

5.1.1. Datensätze

In dieser Arbeit werden die Datensätze CM1-NASA, Dronology, GANNT, Modis und WARC, welche von Hey u. a. [21] genutzt wurden, verwendet. Einerseits sind die TLs von den HLRs zu den LLRs enthalten, was eine notwendige Voraussetzung ist. Andererseits bieten die Datensätze den Vorteil, dass Hey u. a. [21] auf den Datensätzen die benötigten Experimente für PA1 schon durchführten, wodurch weniger Ressourcen für diese Bachelorarbeit verbraucht werden. Zusätzlich decken die Datensätze verschiedene Domänen ab.

Die Anzahl der Artefakte, der daraus resultierenden Kombinationen und der TLs sind in Tabelle 5.1 dargestellt. Dronology ist der größte und Modis der kleinste Datensatz. CM1-NASA und GANNT haben eine ähnliche Größe wie Modis. WARC hat eine mittlere Größe und ist etwa sechsmal so groß wie Modis. Auch der prozentuale Anteil an TLs ist bei CM1-NASA, GANNT und Modis ähnlich. Bei WARC ist dieser Anteil kleiner und bei Dronology noch geringer.

Wenn die Anzahl der Wörter in den Artefakten betrachtet wird, welche in Abbildung 5.1 dargestellt sind, dann wird folgendes sichtbar: Die Wörteranzahlen in den Anforderungen sind in einem ähnlichen Bereich mit Ausnahme der Anzahlen in den LLRs in CM1-NASA, welche sowohl im Median als auch im Maximum deutlich größer sind. Die gleiche Beobachtung lässt sich auch bei der Anzahl an BERT- und GPT-Tokens machen, welche in Abbildungen des Anhangs dargestellt sind.

5.1.2. Hyperparameteroptimierung

Für alle Experimente mit dem FA werden Hyperparameter für das Training benötigt. Üblicherweise werden diese für jedes Modell einzeln ermittelt [6]. Die Trainingszeit und der Ressourcenverbrauch würden in dieser Bachelorarbeit dabei aber stark erhöht werden, da für diese Bachelorarbeit sehr viele verschiedene Modelle trainiert werden. Dies würde den Rahmen der Bachelorarbeit überschreiten. Aus diesem Grund wird die Hyperparameteroptimierung nur einmal vor allen Experimenten mit dem FA durchgeführt und es werden die gleichen optimalen Hyperparameter für alle Modelle in allen Experimenten genutzt.

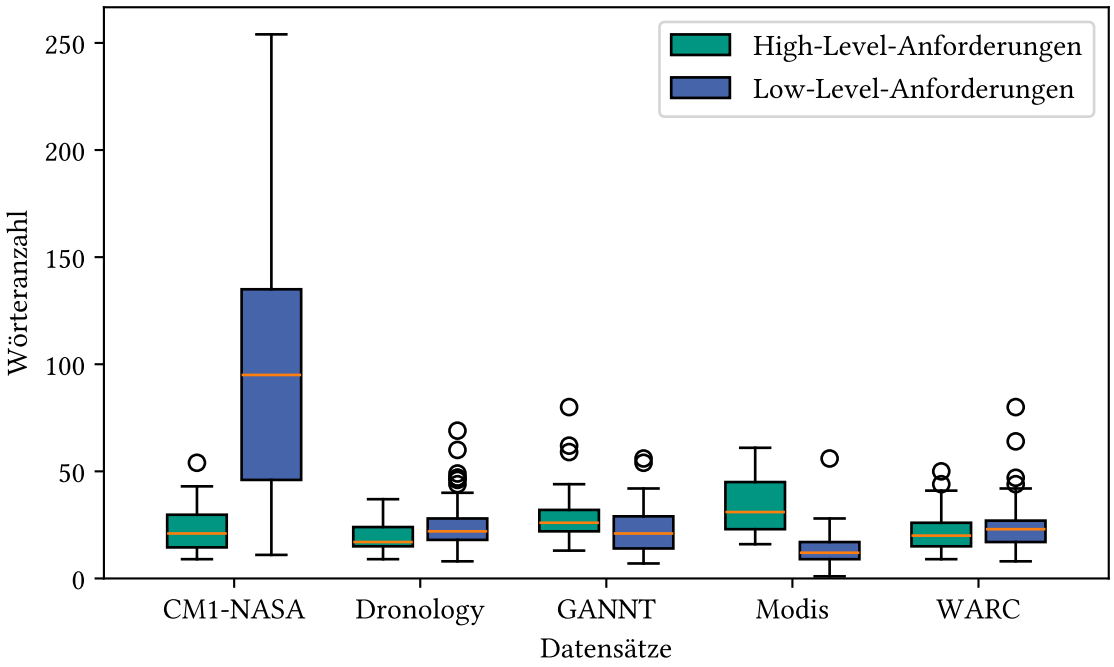

Abbildung 5.1.: Anzahl der Wörter in den Anforderungen der Datensätze

Für die Hyperparameteroptimierung wird die Strategie *Grid Search* verwendet. Als Datenaufteilungsstrategie wird *mixed-projekt*, welche in Abbildung 5.2 dargestellt ist, genutzt. Bei dieser Strategie werden alle Daten aus allen Projekten zusammengesetzt und durchmischt. Danach wird die entstandene Menge in k gleich große Mengen stratifiziert aufgeteilt. Eine Menge davon wird für das Testen verwendet und die anderen Mengen als TBD. Diese Datenaufteilungsstrategie wird in der Hyperparameteroptimierung genutzt, da bei dieser Strategie Daten aus allen Projekten sowohl in der Trainings- als auch in der Testmenge vorliegen, wodurch ein möglichst breites Spektrum an Daten abgedeckt wird. Dabei werden die Daten in fünf Teile ($k = 5$) aufgeteilt, wobei eine Aufteilung für das Testen genutzt wird und die rechtlichen vier für das Training der Modelle. Bei dieser Hyperparameteroptimierung wird keine Kreuzvalidierung durchgeführt. Einerseits würde sich dadurch der Zeitaufwand verfünffachen und andererseits beinhaltet die Testmenge viele und variable Daten, sodass keine großen Varianzen der Performance zwischen den einzelnen Testmengen zu erwarten sind. Bei der Hyperparameteroptimierung werden nur die wichtigen Hyperparameter Anzahl an Trainingsepochen, Lernrate, Gewichtsverfall und Anzahl an Trainingsdaten pro Trainingsschritt pro Gerät optimiert. Alle anderen Hyperparameter werden auf den Standardwerten belassen, welche von der *Trainer*-Klasse der Python-Bibliothek *transformers* vorgegeben werden.

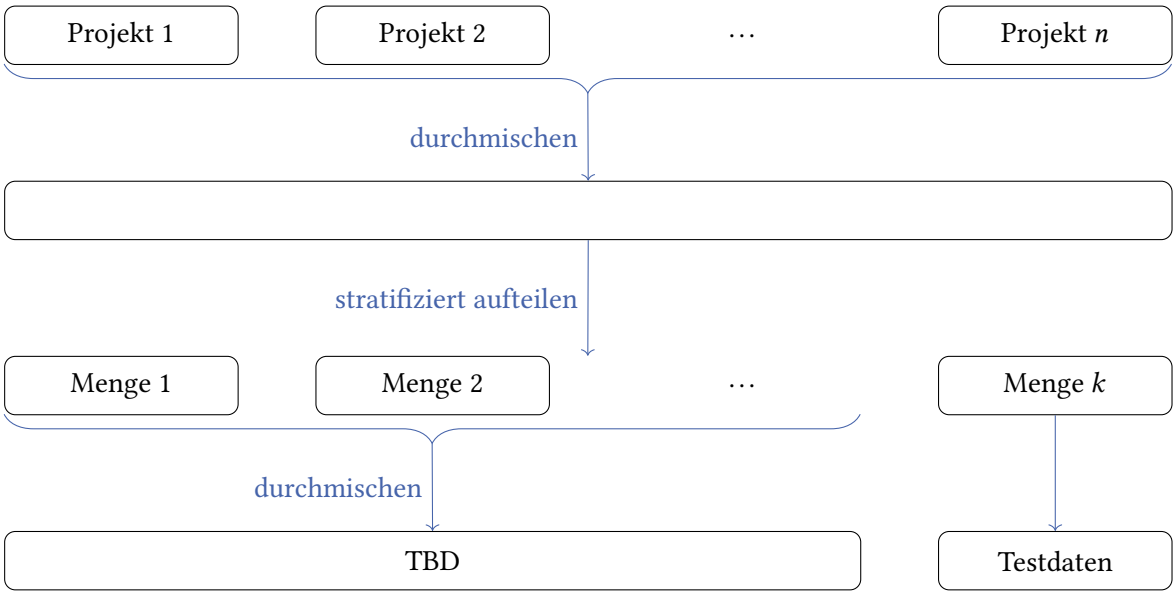

Abbildung 5.2.: *mixed-projekt* Datenaufteilungsstrategie - n entspricht der Anzahl an Projekten - $k \in \mathbb{N}_+$

5.1.2.1. Erste Stufe

Bei der Hyperparameteroptimierung werden vorerst folgende Hyperparameter mit den zugehörigen Werten getestet:

- Anzahl an Trainingsepochen: 10; 20
- Lernrate: 5e-6; 1e-5; 5e-5
- Gewichtsverfall: 0,0
- Anzahl an Trainingsdaten pro Trainingsschritt pro Gerät: 8; 16; 32

Bei der Durchführung war auffällig, dass es bei der Anzahl an Trainingsdaten pro Trainingsschritt pro Gerät 32 öfters zu Abstürzen kam, da der verfügbare Grafikkartenspeicher nicht ausreichte.

Die Ergebnisse (F_1-Werte) des ersten Durchlaufs sind in Abbildung 5.3 dargestellt. Bei der Anzahl an Trainingsepochen wird sichtbar, dass es bei einer höheren Epochenanzahl zu besseren Ergebnissen kommt, wenn die Ausreißer außer Acht gelassen werden. Zusätzlich wird bei der Anzahl an Trainingsdaten pro Trainingsschritt pro Gerät augenfällig, dass die Ergebnisse bei allen genutzten Werten in einem ähnlichen Bereich liegen, wenn die Ausreißer ebenfalls außer Acht gelassen werden. Bei der Lernrate fällt auf, dass die Ausreißer nur bei 5e-5 auftreten. Zusätzlich zeigte sich, dass die Lernrate 1e-5 im Schnitt bessere Ergebnisse lieferte als 5e-6.

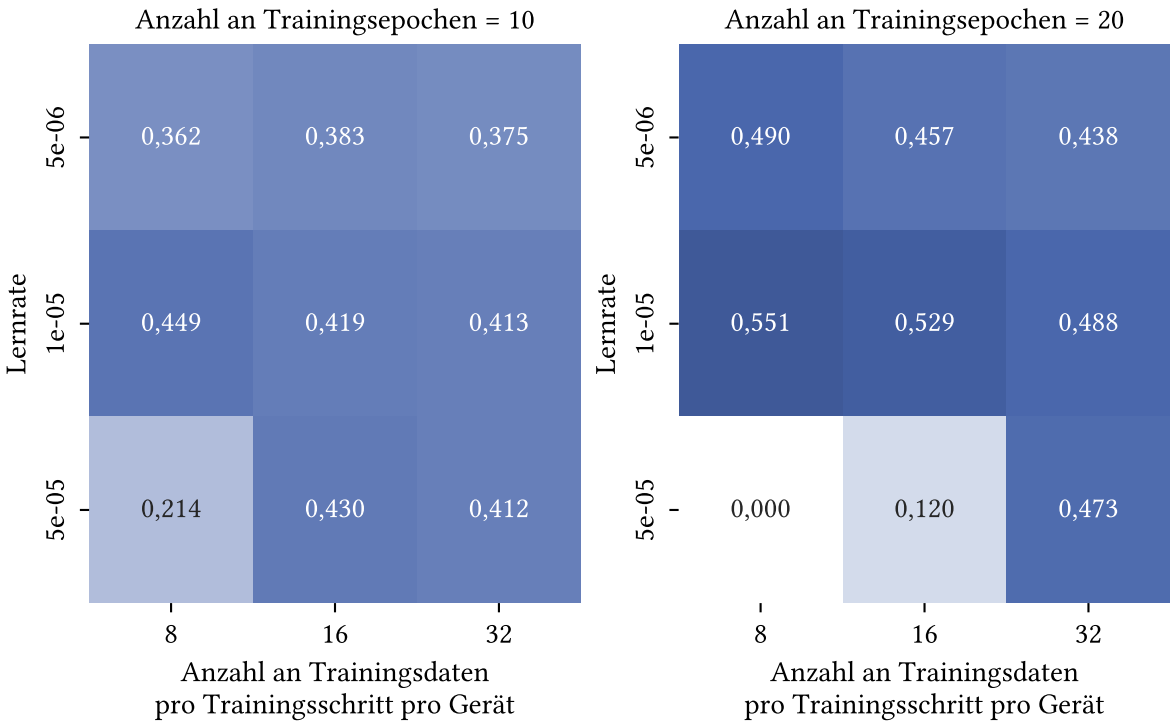

Abbildung 5.3.: Ergebnisse (F_1-Werte) der ersten Stufe der Hyperparameteroptimierung

5.1.2.2. Zweite Stufe

Um zu überprüfen, ob die Leistung mit einer weiteren Erhöhung der Anzahl an Trainingsepochen gesteigert werden kann, wird eine zweite Stufe der Hyperparameteroptimierung durchgeführt. Dabei wird sowohl mit der Anzahl an Trainingsepochen als auch mit dem Gewichtsverfall variiert. Der Gewichtsverfall wird erst in dieser Stufe variiert, da die Durchführungsdauer der ersten Stufe zu groß gewesen wäre. Die Lernrate und die Anzahl an Trainingsdaten pro Trainingsschritt pro Gerät wird in dieser Stufe festgesetzt, da die Varianzen mit Ausnahme der Ausreißer gering ausfielen und eine Variation der Werte den Zeitbedarf deutlich erhöhen würde. Die Lernrate wird auf 1e-5 festgesetzt, da mit dieser Konfiguration im Schnitt die beste Leistung in der vorherigen Stufe erzielt werden konnte. Für die Anzahl an Trainingsdaten pro Trainingsschritt pro Gerät wird 16 festgelegt, da 16 im Schnitt bessere Leistung als acht brachte und da 32 zu Abstürzen des Testprogramms führte.

Für diese Stufe der Hyperparameteroptimierung werden folgende Hyperparameter mit den zugehörigen Werten genutzt:

- Anzahl an Trainingsepochen: 16; 32; 64; 128
- Lernrate: 1e-5
- Gewichtsverfall: 0,0; 0,1; 0,2; 0,3; 0,4
- Anzahl an Trainingsdaten pro Trainingsschritt pro Gerät: 16

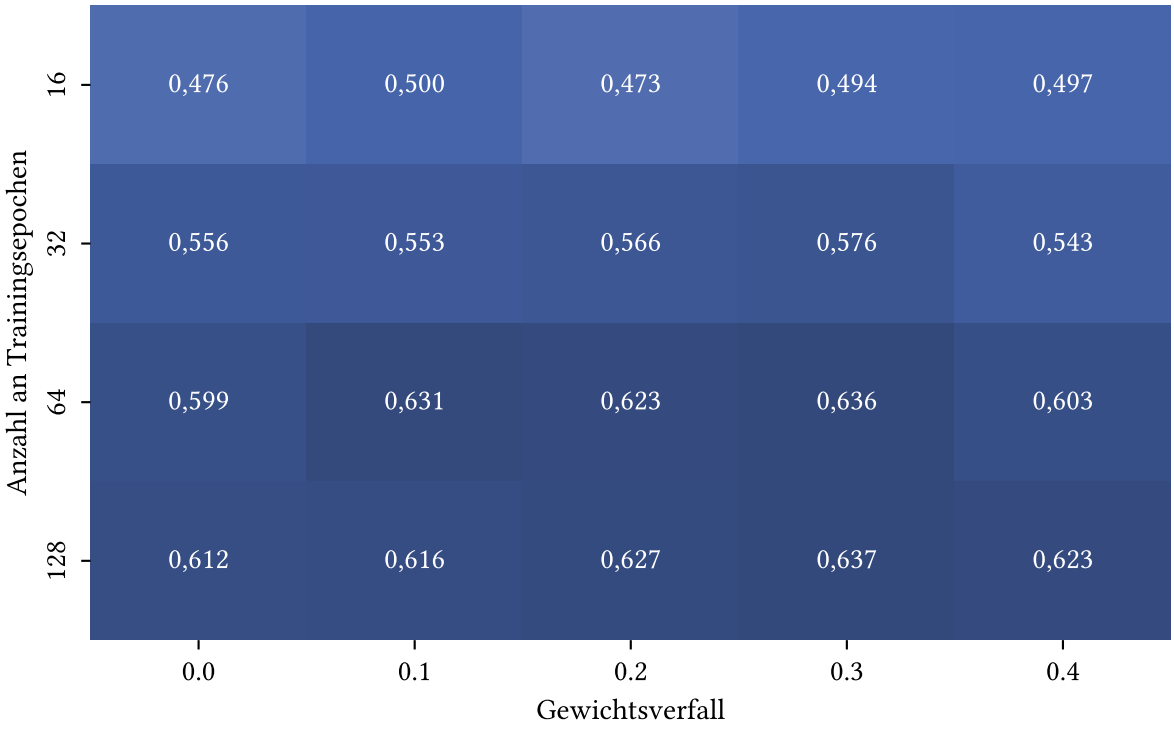

Abbildung 5.4.: Ergebnisse (F_1-Werte) der zweiten Stufe der Hyperparameteroptimierung

Bei der eigentlichen Durchführung der Hyperparameteroptimierung gab es keine Auffälligkeiten.

Die Ergebnisse (F_1-Werte) dieser Stufe sind in Abbildung 5.4 dargestellt. Eine Steigerung der Leistung mit Erhöhung der Anzahl an Trainingsepochen zeigte sich erneut, wobei aber sichtbar wird, dass diese Steigerung mit der Erhöhung der Anzahl an Trainingsepochen immer geringer wird. Zusätzlich wird augenfällig, dass die Variation des Gewichtsverfalls bei fester Anzahl an Trainingsepochen keinen großen Einfluss auf die Performance hat. Die beste Leistung brachte die Kombinationen aus dem Gewichtsverfall 0,3 und der Anzahl an Trainingsepochen 128.

Es werden keine starken Leistungsverbesserungen mit weiterer Steigerung der Anzahl an Trainingsepochen erwartet, da sich zeigte, dass die Performance mit Erhöhung nicht mehr stark steigt. Zusätzlich würde eine weitere Erhöhung die Trainingszeit noch weiter erhöhen. Aus diesem Grund werden nun die optimierten Hyperparameter für die Experimente wie folgt festgelegt:

- Anzahl an Trainingsepochen: 128
- Lernrate: 1e-5
- Gewichtsverfall: 0,3
- Anzahl an Trainingsdaten pro Trainingsschritt pro Gerät: 16

5.2. Szenario 1: TL-Generierung

In diesem Szenario stehen dem Entwickler keine früheren Projekte mit dokumentierten TLs zur Verfügung. Zusätzlich sind keine TLs im aktuellen Projekt, auf welchem die TLR durchgeführt wird, dokumentiert. Zudem beabsichtigt er nicht, eigene Ressourcen für eine selbstständige Annotation dieser TLs einzusetzen. Ziel des Entwicklers ist es, alle TLs des aktuellen Projekts zu ermitteln. Damit wird in diesem Szenario die TLR-Aufgabe TL-Generierung bearbeitet.

Für dieses Szenario fällt der FA raus, da er zwangsweise annotierte Daten für die Feinanpassung benötigt. Eine Verwendung ohne die Feinanpassung ergibt keinen Sinn, da die Klassifikationsschicht des genutzten Modells neu initialisierte Modellparameter beinhaltet, wodurch die Klassifikation zufällig wäre [24]. Auch der PA3 kann in diesem Szenario nicht genutzt werden, da keine annotierten Daten für die Beispiele vorliegen. PA1 und PA2 sind für dieses Szenario geeignet, da diese Ansätze keine annotierten Daten benötigen. Für PA1 liegen die Ergebnisse bereits vor und für PA2 stellen mir meine Betreuer die Ergebnisse zur Verfügung.

Da in diesem Szenario keine Feinanpassung infrage kommt, entfällt ein direkter Vergleich zwischen Feinanpassung und Prompting. Dennoch wird ein Experiment und eine Auswertung durchgeführt, da Experiment 1 neue Ergebnisse liefert, welche von der Forschungsgemeinschaft noch nicht betrachtet wurden.

Aus dem Vorherigen ergibt sich für dieses Szenario die **Forschungsfrage 1**: Welchen Einfluss hat die Verwendung von RAG auf die Leistung bei der automatisierten TL-Generierung von HLRs zu LLRs mit *zero-shot* Prompting?

5.2.1. Experiment 1: *zero-shot* Prompting ohne Retrieval-Augmented Generation (Prompting-Ansatz 2)

In diesem Experiment wird der PA2 verwendet, indem das LiSSA-Framework [11] für den Ansatz angepasst wird. Eine DA ist nicht notwendig, da keine TBD aus den Datensätzen entnommen werden, wodurch auf allen Daten getestet wird. Der Zufallswert ist hierbei auf 133742243 festgesetzt.

Die Ergebnisse des Experiments sind in Tabelle 5.2 dargestellt. Die Präzision liegt bei allen Testdatensätzen in einem niedrigen Bereich unter oder nahe 0,3 und die Ausbeute ist bei allen Testdatensätzen sehr hoch mit Ausnahme von Modis. Bei Modis liegt die Ausbeute in einem ähnlichen Bereich wie die Präzision. Die ermittelten F_1-Werte liegen alle um 0,3 und die F_2-Werte in einem Bereich von 0,35 bis 0,6.

	zero-shot Prompting ohne RAG (PA2)				*zero-shot* Prompting mit RAG (PA1)	
Testdatensatz	Präzision	Ausbeute	F_1	F_2	F_1	F_2
CM1-NASA	0,240	0,933	0,382	**0,592**	**0,519**	0,565
Dronology	0,141	0,900	0,244	0,434	**0,575**	**0,620**
GANNT	0,206	0,926	0,337	0,545	**0,574**	**0,556**
Modis	0,319	0,366	**0,341**	**0,355**	0,255	0,197
WARC	0,168	0,912	0,284	0,484	**0,584**	**0,616**
Durchschnitt	0,215	0,807	0,318	0,482	**0,501**	**0,511**

Tabelle 5.2.: Ergebnisse von Experiment 1 (*zero-shot* Prompting ohne RAG (Prompting-Ansatz 2)) und *zero-shot* Prompting mit RAG (Prompting-Ansatz 1)

5.2.2. Auswertung

Die Ergebnisse für die Auswertung dieses Szenarios und die Beantwortung der Forschungsfrage 1 sind ebenfalls in Tabelle 5.2 aufgezeigt. Wenn man die Ansätze miteinander vergleicht, dann wird sichtbar, dass die Verwendung von RAG bei PA1 einen deutlichen Zuwachs an Leistung bringt. Dieser Zuwachs zeigt sich besonders bei den F_1-Werten, die im Durchschnitt um 0,183 höher liegen. Auch bei den F_2-Werten ist im Schnitt ein Zuwachs erkennbar, welcher aber deutlich geringer ausfällt. Modis und CM1-NASA sind die einzigen Datensätze, die herausstechen. Der *zero-shot* Prompting-Ansatz ohne RAG (PA2) liefert nur bei Modis bei den F_1- und F_2-Werten und bei CM1-NASA bei dem F_2-Wert eine bessere Performance.

Die **Forschungsfrage 1** kann mit diesen Ergebnissen wie folgt beantwortet werden: Die Verwendung von RAG verbessert die Leistung bei der automatisierten TL-Generierung mit *zero-shot* Prompting. Dies zeigt sich besonders bei den F_1-Werten, welche für die vollautomatisierte TLR am wichtigsten sind [11]. Der Ansatz *zero-shot* Prompting mit RAG bringt die beste Performance in diesem Szenario.

5.3. Szenario 2: TL-Generierung mit optionalem Wissenstransfer

In diesem Szenario stehen dem Entwickler nur frühere Projekte mit dokumentierten TLs zur Verfügung. Es sind keine TLs im aktuellen Projekt, auf welchem die TLR durchgeführt wird, dokumentiert und er beabsichtigt nicht, eigene Ressourcen für eine selbstständige Annotation dieser TLs einzusetzen. Ziel des Entwicklers ist es, alle TLs des aktuellen Projekts zu ermitteln. Damit wird in diesem Szenario die TLR-Aufgabe TL-Generierung bearbeitet.

In diesem Szenario sind alle Ansätze geeignet, da annotierte Daten vorliegen. Die Ergebnisse von PA1 und PA2 liegen bereits vor und die Performance vom FA wird in Experiment 2

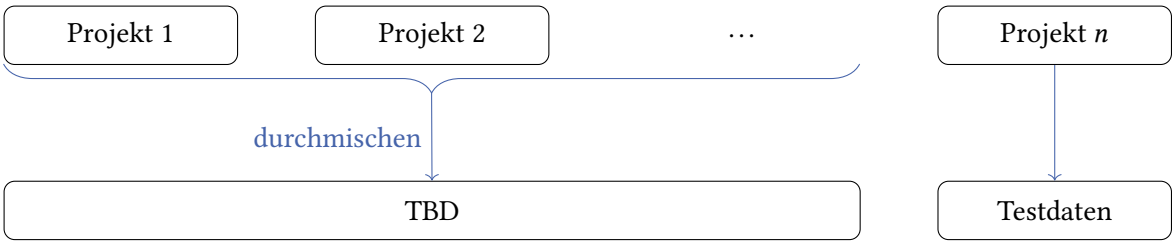

Abbildung 5.5.: *cross-projekt* Datenaufteilungsstrategie - *n* entspricht der Anzahl an Projekten

ermittelt. Mit PA3 wird für dieses Szenario kein Experiment durchgeführt, da die verfügbaren Ressourcen dieser Bachelorarbeit nur für maximal ein Experiment mit PA3 ausreichen und dieses Experiment in Szenario 3 durchgeführt wird, indem projekt-interne Daten vorliegen.

Für das Szenario und das zugehörige Experiment wird folgende **Forschungsfrage 2** gestellt: Welchen Einfluss hat Wissenstransfer aus anderen Projekten auf die Leistung bei der automatisierten TL-Generierung von HLRs zu LLRs und wie schneiden Feinanpassung und *zero-shot* Prompting in diesem Kontext im Vergleich ab?

5.3.1. Experiment 2: Feinanpassungsansatz mit *cross-projekt* Datenaufteilung

Bei diesem Experiment wird der FA in Kombination mit der *cross-projekt* DA genutzt, welche in Abbildung 5.5 aufgezeigt ist. Hierbei werden alle Daten eines Projektes zum Testen verwendet und alle anderen Daten aus den anderen Projekten als TBD genutzt. In diesem Experiment werden alle fünf vorhandenen Datensätze mit einbezogen. Dabei wird Kreuzvalidierung verwendet, wodurch jeder Datensatz einmal zum Testen genutzt wird.

Die Ergebnisse dieses Experiments sind in Tabelle 5.3 dargestellt. Sowohl die Präzision als auch die Ausbeute liegen in einem großen Bereich. Die Präzision geht von 0,177 bis 1 und die Ausbeute deckt einen Bereich von 0,178 bis 0,773 ab. Bei CM1-NASA, GANNT und Modis ist die Präzision deutlich höher als die Ausbeute. Dieser Unterschied fällt bei Modis, wo die Präzision 1 und Ausbeute 0,049 beträgt, am größten aus. Bei Dronology hingegen ist die Ausbeute deutlich größer als die Präzision und bei WARC liegen die beiden Werte in einem ähnlichen Bereich zwischen 30 % und 40 %. Die F_1-Werte liegen bei allen Datensätzen um 0,3, mit Ausnahme von Modis, wo ein niedriger F_1-Wert von 0,093 erreicht wird. Bei Modis ist der F_2-Wert noch niedriger als der F_1-Wert. Die F_2-Werte der anderen Datensätze liegen zwischen 0,204 und 0,463. Zusätzlich zeigen die Ergebnisse, dass eine größere Trainingsdatenanzahl in diesem Fall nicht zwangsweise die Performance verbessert. Dies wird bei Dronology und WARC deutlich, für die weniger Trainingsdaten zur Verfügung standen, da sie verglichen mit den anderen Datensätzen größer sind.

Testdatensatz	Präzision	Ausbeute	F_1	F_2
CM1-NASA	0,500	0,178	0,262	0,204
Dronology	0,177	0,773	0,288	0,462
GANNT	0,455	0,294	0,357	0,316
Modis	1,000	0,049	0,093	0,060
WARC	0,377	0,316	0,344	0,327
Durchschnitt	0,502	0,322	0,269	0,274

Tabelle 5.3.: Ergebnisse von Experiment 2 (Feinanpassungsansatz mit *cross-projekt* Datenaufteilung)

	zero-shot Prompting mit RAG (PA1)		*zero-shot* Prompting ohne RAG (PA2)		Feinanpassungsansatz mit *cross-projekt* Datenaufteilung	
Testdatensatz	F_1	F_2	F_1	F_2	F_1	F_2
CM1-NASA	**0,519**	0,565	0,382	**0,592**	0,262	0,204
Dronology	**0,575**	**0,620**	0,244	0,434	0,288	0,462
GANNT	**0,574**	**0,556**	0,337	0,545	0,357	0,316
Modis	0,255	0,197	**0,341**	**0,355**	0,093	0,060
WARC	**0,584**	**0,616**	0,284	0,484	0,344	0,327
Durchschnitt	**0,501**	**0,511**	0,318	0,482	0,269	0,274

Tabelle 5.4.: Ergebnisse für die Auswertung von Szenario 2 (TL-Generierung mit optionalem Wissenstransfer)

5.3.2. Auswertung

Für die Auswertung wird sich auf die Einordnung der Ergebnisse des neu durchgeführten Experiments 2 in die vorhandenen Ergebnisse von PA1 und PA2 beschränkt, da ein Vergleich zwischen PA1 und PA2 bereits in Szenario 1 durchgeführt wird.

Die Ergebnisse für die Auswertung sind in Tabelle 5.4 dargestellt. Bei allen Datensätzen liefert der FA schlechtere Ergebnisse als der PA1. Verglichen mit dem PA2 liefert der FA teilweise bessere Ergebnisse. Dies zeigt sich sowohl bei Dronology (F_1- und F_2-Werte) als auch bei GANNT und WARC (F_1-Werte). Da bei Modis die Performance deutlich schlechter war, ist der FA auch im Schnitt schlechter als der PA2, was sich hauptsächlich bei den F_1-Werten zeigt.

Die **Forschungsfrage 2** kann demnach wie folgt beantwortet werden: Durch Wissenstransfer aus anderen Projekten kann bei der TL-Generierung von HLRs zu LLRs mit Feinanpassung keine bessere Leistung wie *zero-shot* Prompting erzielt werden. Einfaches *zero-shot* Prompting ohne RAG liefert ähnliche aber leicht bessere Ergebnisse wie Feinanpassung mit projekt-externen Daten. *Zero-shot* Prompting mit RAG liefert deutlich bessere Ergebnisse wie Feinanpassung und ist in diesem Szenario der Ansatz mit der besten Performance.

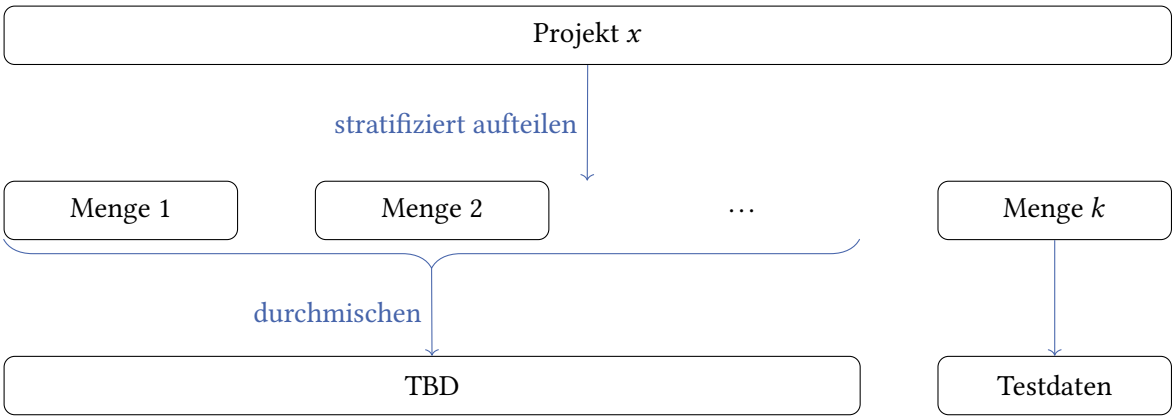

Abbildung 5.6.: *intra-projekt* Datenaufteilungsstrategie - $x \in \{1, 2, \ldots, n \mid n$ entspricht der Anzahl an Projekten$\}$ - $k \in \mathbb{N}_+$

5.4. Szenario 3: TL-Vervollständigung

In diesem Szenario stehen dem Entwickler keine früheren Projekte mit dokumentierten TLs zur Verfügung. Im aktuellen Projekt, auf welchem die TLR durchgeführt wird, stehen einige TLs zur Verfügung bzw. beabsichtigt der Entwickler eigene Ressourcen für eine selbstständige Annotation dieser TLs einzusetzen. Ziel des Entwicklers ist es, die restlichen TLs des aktuellen Projekts zu ermitteln. Damit wird in diesem Szenario die TLR-Aufgabe TL-Vervollständigung bearbeitet.

In diesem Szenario sind alle Ansätze geeignet, da annotierte Daten vorliegen. Die Ergebnisse von PA1 und PA2 liegen bereits vor und die Performance von PA3 wird in Experiment 4 ermittelt. Für den FA wird für dieses Szenario ein neues Experiment (Experiment 3) durchgeführt, da die genutzten Trainingsdaten bei Experiment 2 in diesem Szenario nicht verfügbar und projekt-interne Trainingsdaten nun nutzbar sind.

In diesem Szenario wird folgende **Forschungsfrage 3** beantwortet: Welchen Einfluss hat die Menge an vorhandenen projekt-internen TLs auf die Leistung bei der automatisierten TL-Vervollständigung von HLRs zu LLRs und wie schneiden Feinanpassung und Prompting in diesem Kontext im Vergleich ab?

5.4.1. Experiment 3: Feinanpassungsansatz mit *intra-projekt* Datenaufteilung

Bei diesem Experiment wird der FA in Kombination mit der *intra-projekt* DA, welche in Abbildung 5.6 vorgestellt wird, genutzt. Diese Datenaufteilungsstrategie wird für jedes Projekt einzeln ausgeführt. Alle Daten aus einem Projekt werden hierbei stratifiziert in k gleich große Mengen aufgeteilt. Eine Menge davon wird als Testdaten genutzt und die anderen Mengen als TBD. Bei diesem Experiment wird jeder Datensatz stratifiziert in fünf Mengen ($k = 5$) geteilt. In diesem Experiment wird Kreuzvalidierung genutzt, wodurch in jedem Datensatz jede Menge zum Testen verwendet wird. Wie in Abbildung 5.7 dargestellt, werden aus der Trainingsdatenmenge nicht immer alle Daten entnommen, sondern nur

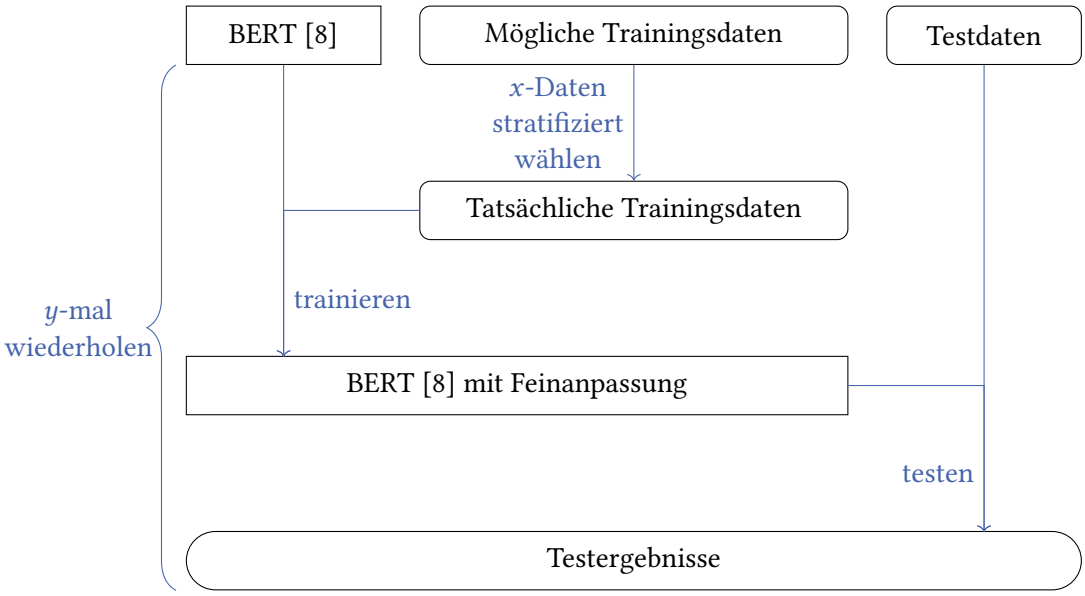

Abbildung 5.7.: Ablauf von Experiment 3 (Feinanpassungsansatz mit *intra-projekt* Datenaufteilung) - Ablauf wird für jede Kombination aus Datensatz, Testmenge und Trainingsdatenanzahl wiederholt - $x \in \{z \in \{32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384, \text{maximum}\} \mid z\text{-Daten sind möglich}\}$ - $y = 4$

eine bestimmte Anzahl stratifiziert. Je Kombination aus Datensatz, Testmenge und Trainingsdatenanzahl werden vier Modelle erstellt. Für jedes dieser Modelle werden zufällig andere Trainingsdaten ausgewählt. Insgesamt werden demnach 20 Modelle pro Trainingsdatenanzahl in jedem Datensatz trainiert und getestet. Bei diesem Experiment wird mit 32 Trainingsdaten gestartet, da bei dieser Anzahl bei den meisten Datensätzen mindestens ein TL vorliegt. Die Trainingsdatenanzahl wächst dann exponentiell und endet mit der größtmöglichen Anzahl, also 80 % der Projektdaten. Der exponentielle Anstieg stellt sicher, dass kleine Datensätze gut abgedeckt werden und die Experimentdauer bei großen Datensätzen im vorgegebenen Rahmen bleibt.

Die Ergebnisse (Durchschnitt und Standardabweichung der F_1- und F_2-Werte pro Trainingsdatenanzahl auf jedem Datensatz) sind in Abbildung 5.8 abgebildet. Die konkreten Werte können in Tabellen des Anhangs eingesehen werden. Sie zeigen, dass sich die F_1- und F_2-Werte nur wenig unterscheiden. Zusätzlich wird sichtbar, dass die durchschnittliche Leistung der feinangepassten PLMs bei wenigen Trainingsdaten schlecht ist und sich mit Erhöhung der Trainingsdatenanzahl steigert. Die durchschnittlichen Werte sind bei wenigen Trainingsdaten bei den kleineren Datensätzen besser. Modis sticht dabei besonders heraus, da die Ergebnisse auf dem Modis-Datensatz im Schnitt deutlich über denen der anderen Datensätze liegen. Die F_1- und F_2-Werte erreichen bei Modis und Dronology bei maximalen Trainingsdaten ähnliche Durchschnittswerte um 0,7. Bei Modis ist die durchschnittliche Abweichung der Werte am Anfang am größten und sinkt mit Erhöhung der Trainingsdatenanzahl. Bei allen anderen Datensätzen ist die Standardabweichung bei geringer Trainingsdatenanzahl klein, steigt dann mit Erhöhung, erreicht das Maximum und stagniert oder sinkt leicht mit weiterer Steigerung der Anzahl an Trainingsdaten.

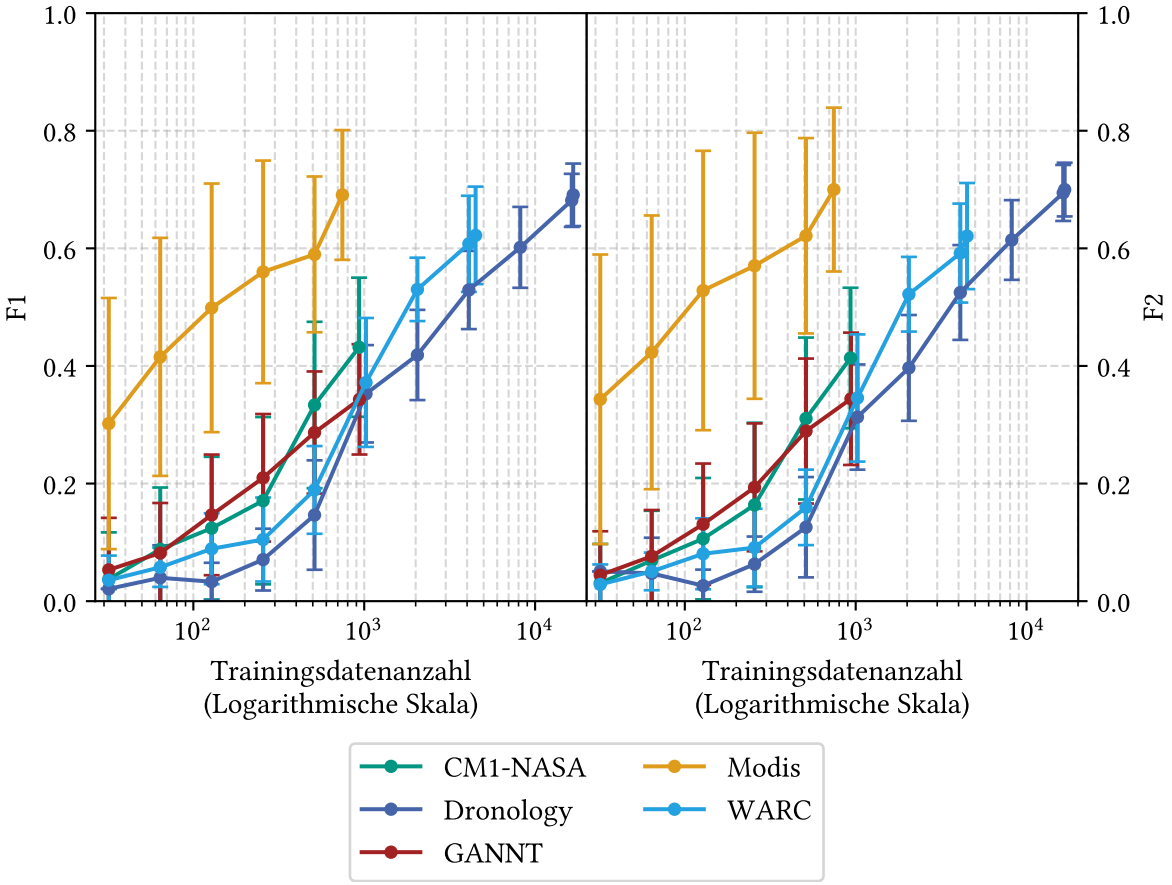

Abbildung 5.8.: Ergebnisse von Experiment 3 (Feinanpassungsansatz mit *intra-projekt* Datenaufteilung) - Durchschnittliche F_1-Werte mit Standardabweichungen (links) und durchschnittliche F_2-Werte mit Standardabweichungen (rechts)

5.4.2. Experiment 4: *few-shot*/*multi-shot* Prompting (Prompting-Ansatz 3) mit *intra-projekt* Datenaufteilung

Bei diesem Experiment wird der PA3 in Kombination mit der *intra-projekt* Datenaufteilung genutzt. Der Zufallswert wird auf 3426785 festgesetzt. Jeder Datensatz wird, wie in Experiment 3, jeweils stratifiziert in fünf Mengen ($k = 5$) aufgeteilt. Zusätzlich wird ebenfalls Kreuzvalidierung verwendet. Die Durchführung des Experiments ist in Abbildung 5.9 aufgezeigt. Es werden nicht alle Daten aus der Beispieldatenmenge entnommen, sondern nur eine festgelegte Anzahl. Diesmal erfolgt die Entnahme nicht stratifiziert, da bei einer geringen Beispieldatenanzahl keine TLs vorhanden wären. Es wird die gleiche Verteilung wie bei der Entnahme pro Epoche beim FA (50 % TLs und 50 % Nicht-TLs) gewählt, weil kein anderer Referenzwert vorhanden ist, da Etezadi u. a. [9] ihre genutzte Verteilung beim *few-shot* Prompting nicht angaben. Für jede Kombination aus Datensatz, Testmenge und Beispieldatenanzahl wird bei CM1-NASA, GANNT, Modis und WARC viermal auf der Testmenge mit zufällig gewählten, unterschiedlichen Beispielen getestet. Bei Dronology wird diese Anzahl auf zwei reduziert, da die API-Kosten den Rahmen der Bachelorarbeit überschreiten würden. Es werden zehn verschiedene Beispieldatenanzahlen verwendet: zwei, vier, sechs,

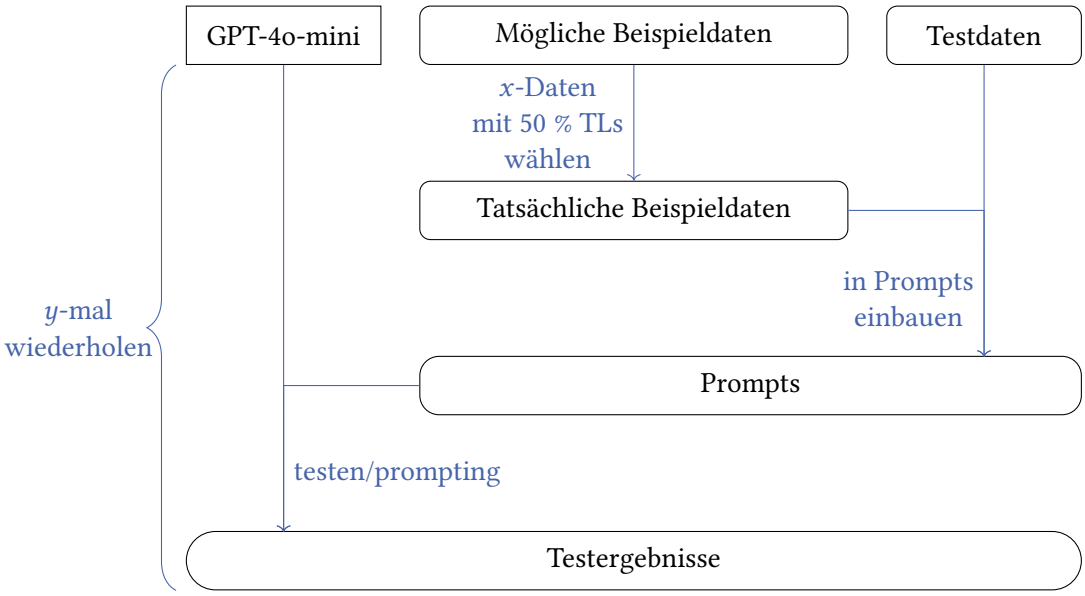

Abbildung 5.9.: Ablauf von Experiment 4 (*few-shot*/*multi-shot* Prompting (PA3) mit *intra-projekt* Datenaufteilung) - Ablauf wird für jede Kombination aus Datensatz, Testmenge und Beispieldatenanzahl wiederholt - $x \in \{2, 4, 6, 8, 10, 20, 30, 40, 50, 60\}$ - $y = 4$ bei CM1-NASA, GANNT, Modis und WARC und $y = 2$ bei Dronology

acht, zehn, 20, 30, 40, 50 und 60. Dadurch wird sowohl der *few-shot* Bereich mit einigen Beispielen als auch der *multi-shot* Bereich mit vielen Beispielen abgedeckt.

Die Ergebnisse des Experiments sind in Abbildung 5.10 veranschaulicht. Zusätzlich sind die genauen Ergebnisse wieder in Tabellen des Anhangs dargestellt. Die durchschnittlichen F_1- und F_2-Werte steigen bei allen Datensätzen mit Steigerung der Beispielanzahl, wobei der Anstieg am Anfang zwischen zwei und zehn Beispielen an größten ist. Danach stagnieren die Werte oder der Anstieg nimmt ab, was am stärksten bei CM1-NASA und Modis sichtbar wird. Insgesamt zeigen die Ergebnisse, dass der Leistungszuwachs insbesondere im Bereich von etwa acht bis zehn Beispieldaten am größten ist, wenn wenige Beispiele genutzt werden. Im Vergleich der Datensätze liefert der Ansatz bei Modis bei den F_1-Werten die besten Ergebnisse und bei Dronology die schlechtesten. Bei CM1-NASA werden ähnliche, jedoch leicht bessere Ergebnisse erzielt als bei Dronology. Die Ergebnisse von GANNT und WARC liegen zwischen den Ergebnissen von Modis und CM1-NASA. Die Leistung, gemessen an den durchschnittlichen F_2-Werten, liegt bei GANNT, Modis und WARC in einem ähnlichen Bereich. Nur bei CM1-NASA und Dronology ist sie deutlich schlechter. Die Standardabweichungen liegen alle unter 0,2 und bei Modis sind sie am größten. Andere Auffälligkeiten sind bei den durchschnittlichen Abweichungen nicht sichtbar.

5.4.3. Auswertung

Für die Auswertung wird sich ebenfalls wieder auf die Einordnung der Ergebnisse der neuen Experimente beschränkt. Bei Ansätzen, bei denen die TBD-Anzahl variiert wird,

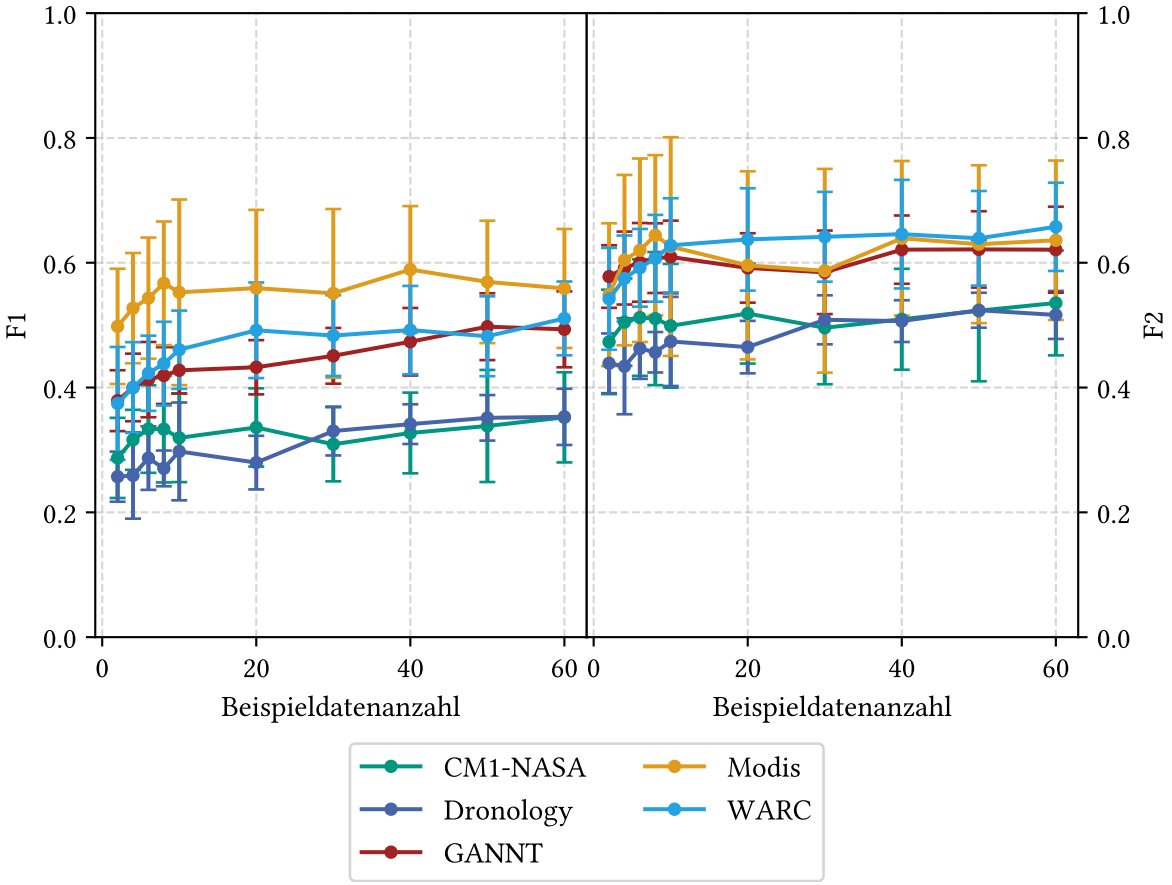

Abbildung 5.10.: Ergebnisse von Experiment 4 (*few-shot*/*multi-shot* Prompting (PA3) mit *intra-projekt* Datenaufteilung) - Durchschnittliche F_1-Werte mit Standardabweichungen (links) und durchschnittliche F_2-Werte mit Standardabweichungen (rechts)

wird in dieser Auswertung der Fokus auf die Anzahl der TLs in den TBD und nicht auf die TBD-Anzahl gelegt. Grund dafür ist, dass die TBD unterschiedliche Verteilung von TL zu Nicht-TL besitzen.

Die durchschnittliche/regressierte Performance der Ansätze über alle Datensätze ist in Abbildung 5.11 dargestellt. Die Leistungen der Ansätze auf jedem Datensatz sind in Abbildungen des Anhangs aufgezeigt, wobei aber auch Ansätze dargestellt sind, die in diesem Szenario nicht nutzbar sind. Der FA mit projekt-internen Trainingsdaten liegt bei einer geringen TL-Anzahl unter zehn im Schnitt deutlich unter allen drei Prompting-Ansätzen. Im Bereich um 15 TLs überschreitet der FA in diesem Szenario im Durchschnitt den einfachen *zero-shot* Prompting-Ansatz (PA2), gemessen an den F_1-Werten. Dies entspricht im Durchschnitt ca. 22 % der Projekt-TLs. Die Leistung des FA, gemessen an den F_1-Werten, übersteigt ab ca. 45 TLs in den Trainingsdaten die Performance vom *zero-shot* Prompting-Ansatz mit RAG (PA1), was durchschnittlich etwa 66 % der Projekt-TLs entspricht. Ab dieser Schwelle liegen auch die F_2-Werte des FA ca. auf dem Niveau beider *zero-shot* Prompting-Ansätze. Der FA überschreitet nicht direkt *few-shot*/*multi-shot* Prompting, was daran liegt, dass nur bis maximal 30 TLs in den Beispieldaten getestet wurde. Wenn die Leistung vom PA3 mit weiterer Steigerung

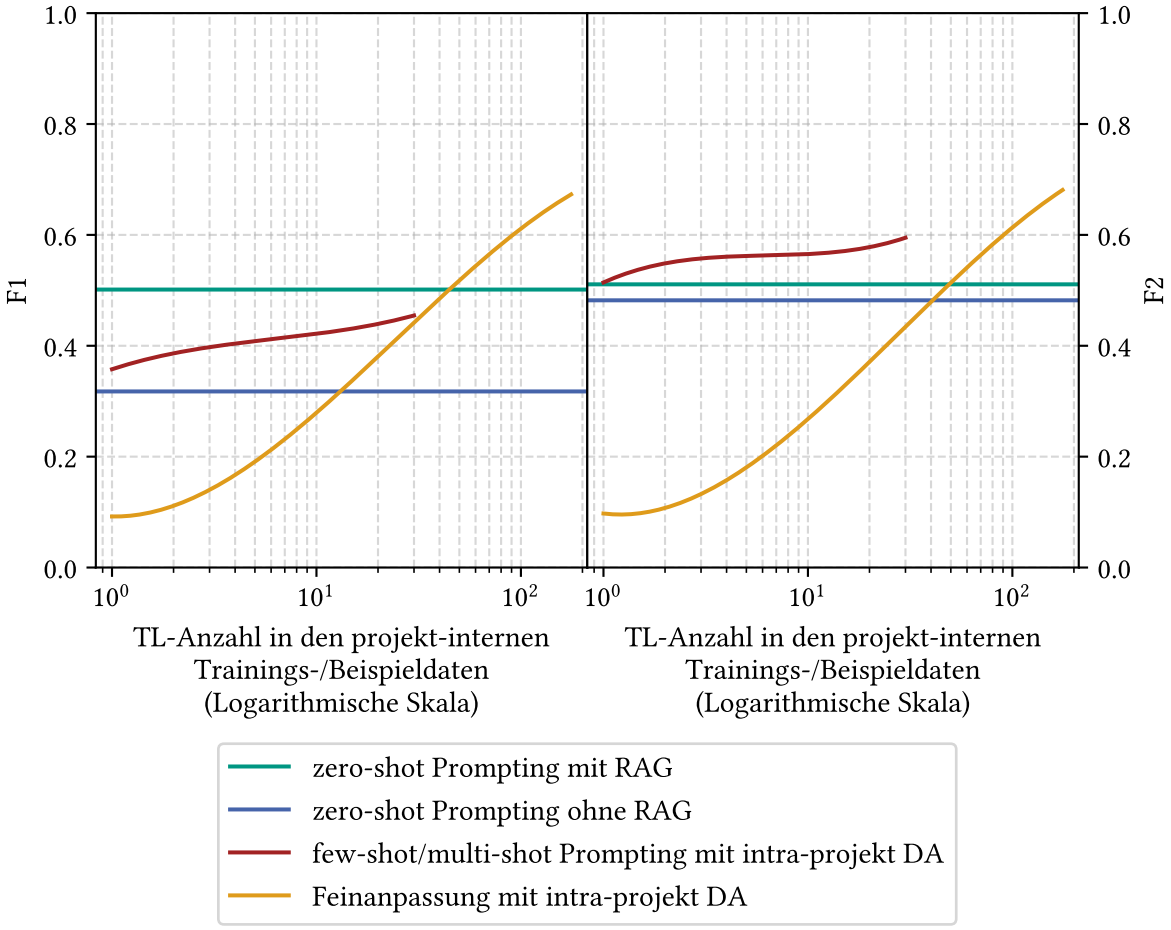

Abbildung 5.11.: Durchschnittliche/regressierte Ergebnisse über alle Datensätze für die Auswertung von Szenario 3 (TL-Vervollständigung) - F_1-Werte (links) und F_2-Werte (rechts) - Spline-Regression (geglättet, log-transformierte x-Werte)

der Beispielanzahl nicht mehr steigt, dann würde der FA *few-shot/multi-shot* Prompting ab ca. 35 TLs (Durchschnittlich ca. 51 % der Projekt-TLs) im F_1-Wert und ab ca. 90 TLs (Durchschnittlich ca. 132 % der Projekt-TLs) im F_2-Wert überschreiten. Die Performance, gemessen an den F_1-Werten, vom *few-shot/multi-shot* Prompting liegt im Schnitt zwischen der Performance vom *zero-shot* Prompting mit RAG und ohne RAG. Hinsichtlich der F_2-Werte übertrifft die durchschnittliche Performance des *few-shot/multi-shot* Promptings bereits ab einem TL in den Beispieldaten die Leistung beider *zero-shot* Prompting-Ansätze.

Die Antwort auf die **Forschungsfrage 3** lässt sich wie folgt zusammenfassen: Wenn sich die Anzahl der TLs in den TBD erhöht, dann steigt auch die Leistung, wenn der Ansatz annotierte Daten benötigt. Die Performance von *few-shot/multi-shot* Prompting ist bei weniger TLs deutlich besser als beim FA, wobei der Anstieg in diesem Szenario deutlich geringer ausfällt als bei der Feinanpassung. Bis zu durchschnittlich etwa 45 vorhandenen TLs aus dem aktuellen Projekt erzielt *zero-shot* Prompting mit RAG die beste Performance für die automatisierte TL-Vervollständigung, während ab einer höheren Anzahl an TLs der FA überlegen ist.

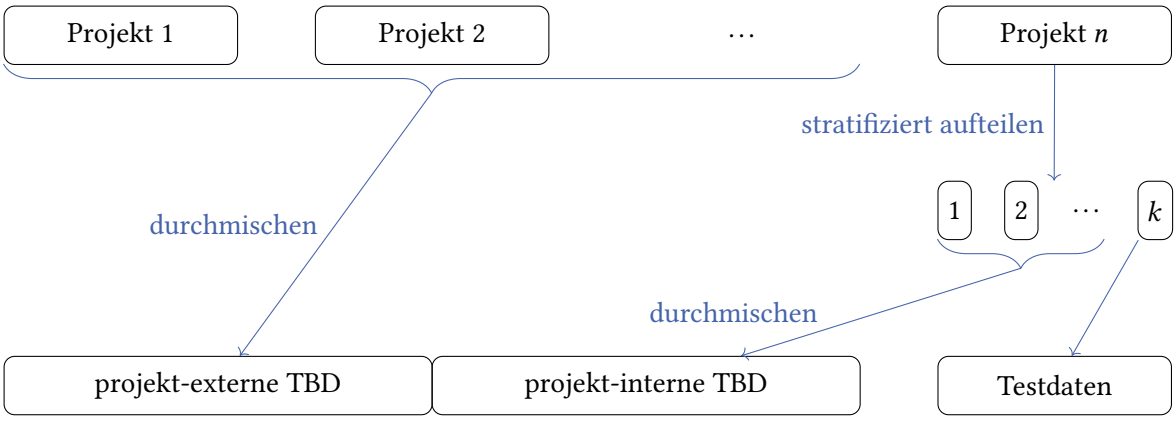

Abbildung 5.12.: *intra-cross-projekt* Datenaufteilungsstrategie - n entspricht der Anzahl an Projekten - $k \in \mathbb{N}_+$

5.5. Szenario 4: TL-Vervollständigung mit optionalem Wissenstransfer

In diesem Szenario stehen dem Entwickler frühere Projekte mit dokumentierten TLs zur Verfügung. Im aktuellen Projekt, auf welchem die TLR durchgeführt wird, stehen einige TLs zur Verfügung bzw. beabsichtigt der Entwickler eigene Ressourcen für eine selbstständige Annotation dieser TLs einzusetzen. Ziel des Entwicklers ist es, die restlichen TLs des aktuellen Projekts zu ermitteln. Damit wird in diesem Szenario die TLR-Aufgabe TL-Vervollständigung bearbeitet.

In diesem Szenario sind alle Ansätze geeignet, da annotierte Daten vorliegen. Zusätzlich sind alle vorherigen Experimente auf dieses Szenario übertragbar, da sowohl projekt-interne als auch projekt-externe Daten vorliegen. Dazu wird für dieses Szenario ein neues Experiment mit dem FA (Experiment 5) durchgeführt, indem projekt-interne und -externe Daten für das Training genutzt werden. Mit PA3 wird für dieses Szenario kein weiteres Experiment durchgeführt, da die verfügbaren Ressourcen dieser Bachelorarbeit nur für maximal ein Experiment, welches in Szenario 3 durchgeführt wird, ausreichen.

Für das Szenario und das zugehörige Experiment wird folgende **Forschungsfrage 4** untersucht: Welchen Einfluss hat Wissenstransfer aus anderen Projekten auf die Leistung bei der automatisierten TL-Vervollständigung von HLRs zu LLRs mit Feinanpassung und wie schneiden Feinanpassung und Prompting in diesem Kontext im Vergleich ab?

5.5.1. Experiment 5: Feinanpassungsansatz mit *intra-cross-projekt* Datenaufteilung

Bei diesem Experiment wird der FA in Kombination mit der *intra-cross-projekt* DA, welche in Abbildung 5.12 veranschaulicht ist, genutzt. Diese Strategie ist eine Kombination aus der *intra-projekt* und der *cross-projekt* Datenaufteilungsstrategie. Bei dieser wird zuerst

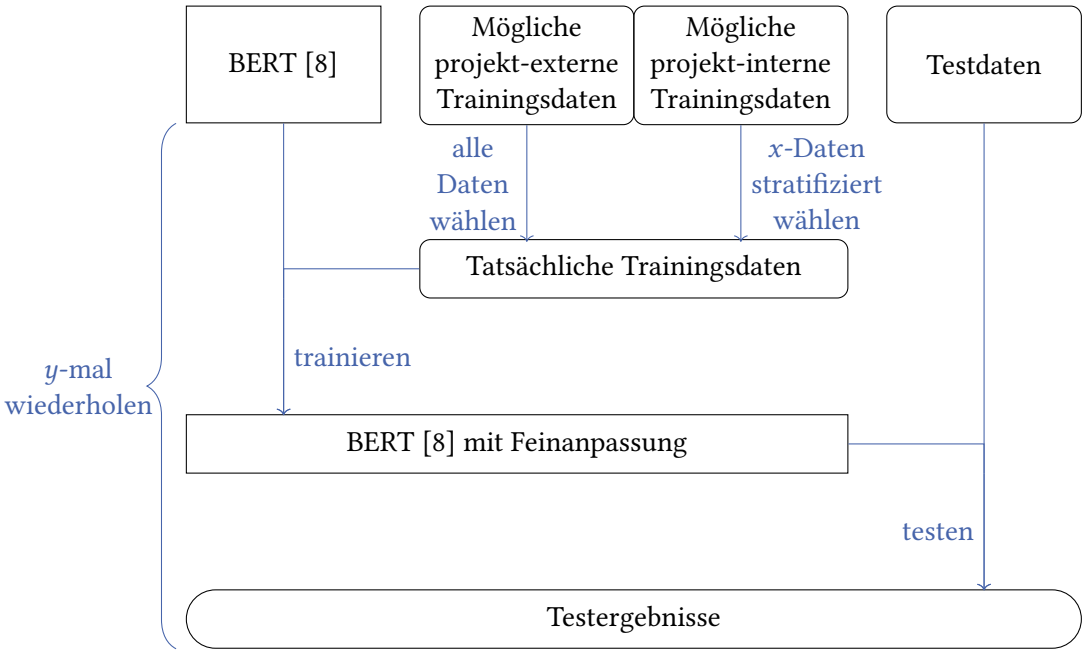

Abbildung 5.13.: Ablauf von Experiment 5 (Feinanpassungsansatz mit *intra-cross-projekt* Datenaufteilung) - Ablauf wiederhole ich für jede Kombination aus Datensatz, Testmenge und Trainingsdatenanzahl - $x \in \{z \in \{32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384, \text{maximum}\} \mid z\text{-Daten sind möglich}\}$ - $y = 2$

ein Projekt nach der *intra-projekt* Datenaufteilungsstrategie aufgeteilt. Dabei erhält man eine Testmenge und eine Menge an TBD. Zu den TBD werden dann noch alle Daten aus den anderen Projekten hinzugefügt. Jeder Datensatz wird bei diesem Experiment wieder jeweils stratifiziert in fünf Mengen ($k = 5$) aufgeteilt. In diesem Experiment wird erneut Kreuzvalidierung durchgeführt. Der Ablauf des Experiments ist in Abbildung 5.13 dargestellt. Zum Training werden alle Daten aus der projekt-externen Trainingsdatenmenge und eine bestimmte Anzahl aus der projekt-internen Trainingsdatenmenge stratifiziert entnommen. Je Kombination aus Datensatz, Testmenge und Trainingsdatenanzahl werden zwei Modelle erstellt. Die projekt-internen Trainingsdatenanzahlen werden gleich wie bei Experiment 3 gewählt.

Die Ergebnisse von Experiment 5 sind in Abbildung 5.14 dargestellt. Dazu ist Performance wieder in Tabellen des Anhangs aufgezeigt. Im Schnitt steigen die Werte bei allen Datensätzen mit Steigerung der Anzahl projekt-interner Trainingsdaten fast dauerhaft mit Ausnahme von Dronology, wo die durchschnittlichen F_2-Werte bis 1024 Trainingsdaten sinken und danach erst steigen. Der größte Anstieg der F_1- und F_2-Werte zeigt sich bei Modis und der kleinste bei GANNT. Die Standardabweichungen bei Dronology sind deutlich geringer verglichen mit den anderen Datensätzen. Zusätzlich schwanken die Standardabweichungen bei den anderen Datensätzen stark.

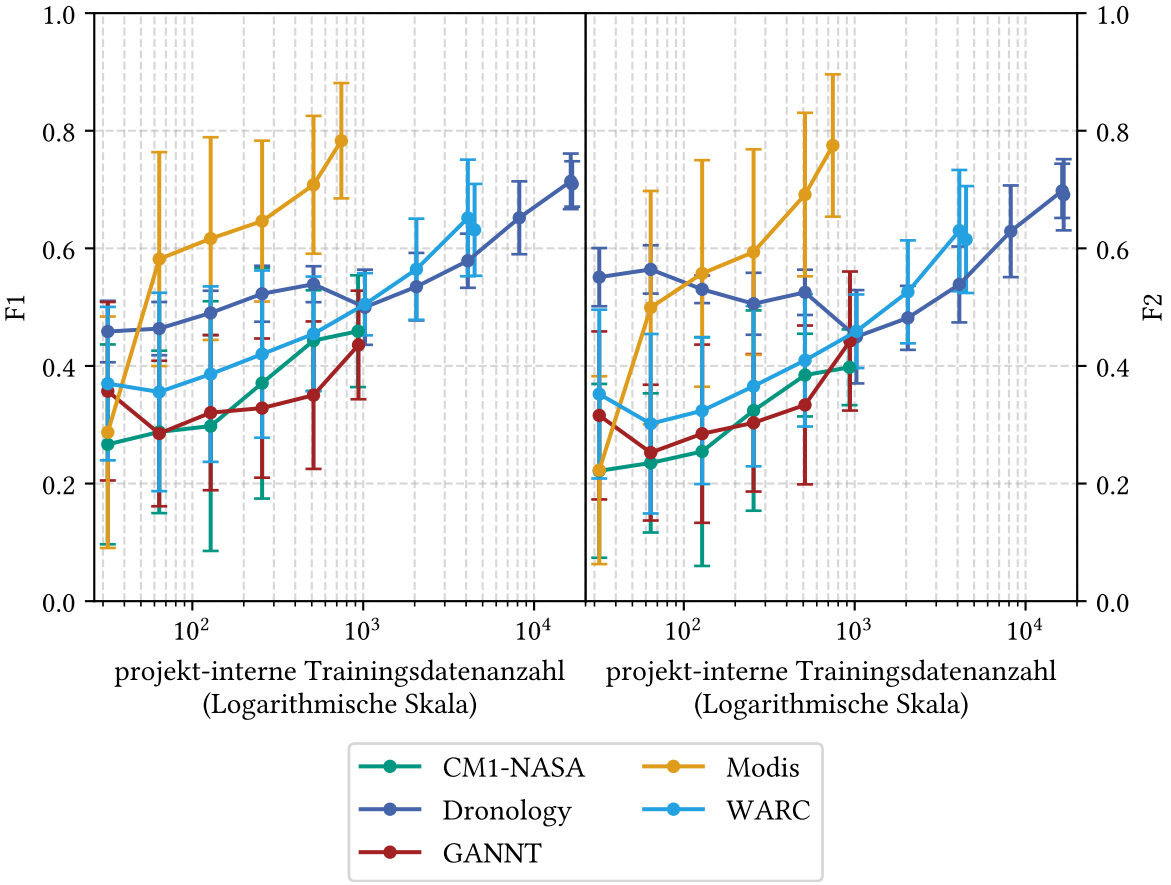

Abbildung 5.14.: Ergebnisse von Experiment 5 (Feinanpassungsansatz mit *intra-cross-projekt* Datenaufteilung) - Durchschnittliche F_1-Werte mit Standardabweichungen (links) und durchschnittliche F_2-Werte mit Standardabweichungen (rechts)

5.5.2. Auswertung

Für die Auswertung wird sich ebenfalls wieder auf Vergleichsmöglichkeiten, die neu hinzukommen, beschränkt. Bei Ansätzen, bei denen die TBD-Anzahl variiert wird, wird in dieser Auswertung der Fokus erneut auf die Anzahl der TLs in den TBD gelegt.

Die durchschnittliche/regressierte Performance der Ansätze über alle Datensätze ist in Abbildung 5.15 aufgezeigt und die Ergebnisse der Ansätze pro Datensatz sind in Abbildungen des Anhangs abgebildet. In Bezug auf die F_1- und F_2-Werte zeigt der FA bereits dann eine bessere Leistung im Durchschnitt, wenn neben projekt-externen Trainingsdaten mindestens ein projekt-interner TL enthalten ist. Zusätzlich fällt die Leistung vom FA, gemessen an den beiden F_β-Werten, im Durchschnitt höher aus, wenn neben projekt-internen auch projekt-externe Daten zum Training genutzt werden. Dieser Leistungszuwachs nimmt mit zunehmender Menge projekt-interner Trainingsdaten ab. Hinsichtlich der F_1-Werte überschreitet der FA mit projekt-internen und -externen Trainingsdaten im Durchschnitt ab ca. zwei projekt-internen TLs (Durchschnittlich ca. 3 % der Projekt-TLs) die Leistung des *few-shot*/*multi-shot* Prompting-Ansatzes und liegt dauerhaft über dem *zero-shot* Prompting

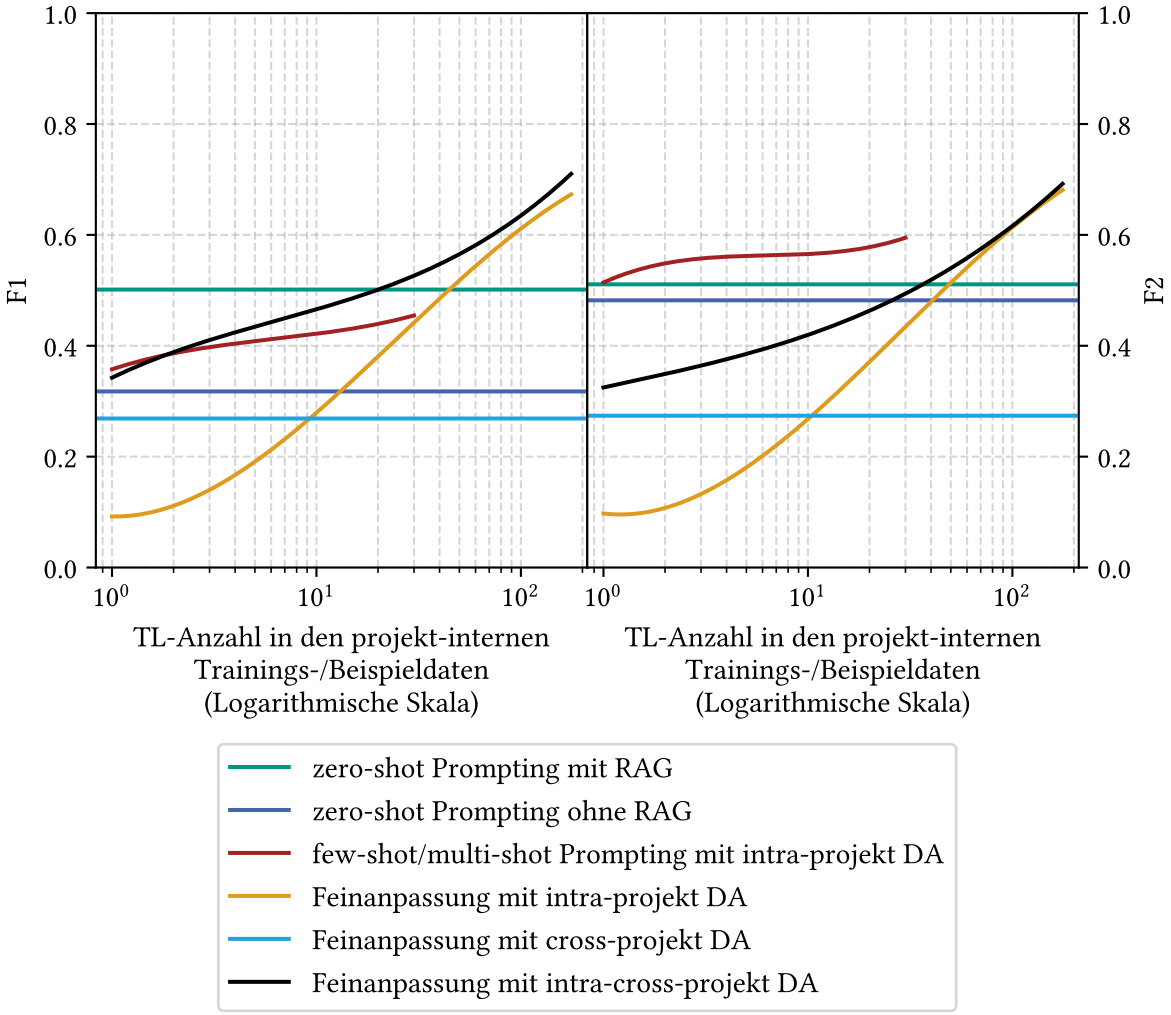

Abbildung 5.15.: Durchschnittliche/regressierte Ergebnisse über alle Datensätze für die Auswertung von Szenario 4 (TL-Vervollständigung mit optionalem Wissenstransfer) - F_1-Werte (links) und F_2-Werte (rechts) - Spline-Regression (geglättet, log-transformierte x-Werte)

Ansatz ohne RAG. Den *zero-shot* Prompting Ansatz mit RAG übertrifft der FA ab durchschnittlich 20 projekt-internen TLs in den Trainingsdaten, gemessen an den F_1-Werten. Dies sind im Schnitt etwa 29 % der Projekt-TLs. Bezüglich der F_2-Werte überschreitet der FA mit projekt-externen und -internen Trainingsdaten *few-shot*/*multi-shot* Prompting (PA3) mit projekt-internen Beispielen nie direkt. Wenn die Leistung vom PA3 mit weiterer Steigerung der Beispielanzahl nicht mehr steigt, dann würde der FA *few-shot*/*multi-shot* Prompting ab ungefähr 90 TLs im F_2-Wert überschreiten. In Bezug auf die F_2-Werte überschreitet der FA mit projekt-internen und -externen Trainingsdaten *zero-shot* Prompting ohne RAG ab ungefähr 15 TLs in den projekt-internen Trainingsdaten und *zero-shot* Prompting mit RAG ab ca. 25 TLs (Durchschnittlich ca. 37 % der Projekt-TLs). Der FA liegt durchschnittlich dauerhaft unter der Leistung von allen Prompting-Ansätzen, wenn nur projekt-externe Trainingsdaten genutzt werden. Außerdem ist dieser Ansatz mit den genutzten Trainingsdaten ab etwa

einer Anzahl von zehn TLs schlechter als der FA mit projekt-internen Trainingsdaten, was im Schnitt ungefähr 15 % der Projekt-TLs sind.

Bezüglich der **Forschungsfrage 4** ergeben sich folgende Erkenntnisse: Die Leistung verbessert sich, wenn man bei der Feinanpassung zusätzlich projekt-externe Daten zu den projekt-internen Trainingsdaten hinzufügt. Diese Verbesserung zeigt sich am stärksten, wenn wenig projekt-interne TLs in den Trainingsdaten vorliegen. Unter 20 TLs im Durchschnitt in den projekt-internen Trainingsdaten liefert der PA1 (*zero-shot* Prompting mit RAG) die beste Performance für die automatisierte TL-Vervollständigung, wenn projekt-externe Daten vorliegen. Ab 20 TLs ist im Schnitt der FA mit projekt-internen und -externen Trainingsdaten der Ansatz mit der besten Leistung.

6. Einschränkungen und Ausblick

In diesem Kapitel werden wichtige Grenzen dieser Bachelorarbeit zusammengefasst. Dabei werden sowohl mögliche Bedrohungen der Validität als auch Limitierungen erläutert. Aus letzteren ergeben sich zugleich Ansatzpunkte für zukünftige Arbeiten.

6.1. Bedrohungen der Validität

Es gibt verschiedene Eigenschaften der durchgeführten Experimente, die die Aussagekraft der Ergebnisse möglicherweise beeinträchtigen. Um die Beeinträchtigungen abzuschwächen, wurden, wenn möglich, verschiedene Maßnahmen unternommen.

6.1.1. Interne Validität

Eine potenzielle Gefährdung der internen Validität ergibt sich durch die Auswahl der TBD und Testdaten, da beobachtete Leistungsunterschiede möglicherweise auf diese Auswahl zurückzuführen sind. Für eine Reduktion dieser möglichen Gefahr wurde die zufällige Wahl der TBD für jede TBD-Anzahl mehrfach durchgeführt. Dabei wurden immer die gleichen Testdaten in Kombination mit Kreuzvalidierung genutzt und die Ergebnisse anschließend gemittelt.

Die Auswahl der Datensätze bedroht unter Umständen ebenfalls die interne Validität [21]. Um diese Bedrohungen zu reduzieren, wurden bekannte Datensätze genutzt, welche von der Forschungsgemeinschaft schon mehrfach verwendet wurden [13, 21].

Eine weitere mögliche Gefährdung der internen Validität stellt die Verwendung von *open-source* Datensätzen dar. Dadurch fand das Vortraining der PLMs möglicherweise auch mit den Daten der Datensätze statt, wodurch die Experimente eventuell nicht die Leistungsfähigkeit der Ansätze ermittelten, sondern nur, ob die PLMs bekannte Daten wiedererkannten.

6.1.2. Externe Validität

Die *open-source* Eigenschaft der Datensätze ist zusätzlich eine Bedrohung der externen Validität, da sich die Ergebnisse möglicherweise nicht auf *closed-source* Datensätze übertragen lassen [21].

Außerdem sind die Datensätze teilweise relativ alt [21]. Dies stellt auch eine Gefährdung der Validität dar, da die Übertragbarkeit der Ergebnisse auf modernere Datensätze möglicherweise eingeschränkt ist [21].

Zusätzlich zeigten Fuchß u. a. [10] und Hey u. a. [21], dass die Datensätze teilweise viele Artefakte enthalten, die keinem TL zugeordnet sind. Diese Eigenschaft kann darauf hindeuten, dass die Datensätze unvollständig sind [10, 21]. Daraus ergibt sich möglicherweise eine Beeinträchtigung der externen Validität, da sich die Ergebnisse gegebenenfalls nicht auf vollständige Datensätze übertragen lassen [24].

Eine weitere potenzielle Bedrohung der Validität ergibt sich durch die Art, wie die Hyperparameteroptimierung durchgeführt wurde. Diese lässt sich nicht direkt auf realistische Anwendungsszenarien übertragen, da bei der durchgeführten Hyperparameteroptimierung alle verfügbaren Daten verwendet wurden, welche in realen Szenarien nicht vorliegen.

Dazu zeigen die genutzten PLMs nichtdeterministisches Verhalten [21, 34]. Um diese eventuelle Bedrohung möglichst weitgehend zu reduzieren, wurde, wenn möglich, ein fester Zufallswert verwendet und die Temperatur auf null gesetzt [21].

6.1.3. Konstruktvalidität

Eine mögliche Bedrohung der Konstruktvalidität ergibt sich durch die Auswahl der Prompts, Modelle und Metriken, da diese Festlegungen die Ergebnisse beeinflussen [11, 18]. Um diese Bedrohung zu verringern, wurden nur Modelle verwendet, die von der Forschung bereits genutzt wurden. Darüber hinaus wurden ausschließlich Prompts genutzt, die entweder in früheren Arbeiten verwendet oder auf Basis dieser abgeleitet wurden. Des Weiteren wurden die Metriken ausgewählt, die am häufigsten von der Forschung für die TLR genutzt wurden.

6.2. Limitierungen und zukünftige Arbeiten

Es gibt verschiedene mögliche Limitierungen dieser Bachelorarbeit, die sich aufgrund begrenzter Ressourcen, wie z.B. Zeit oder Geld, ergaben. Diese Limitierungen bieten Potenzial für zukünftige Arbeiten.

Bei den Untersuchungen wurden die PLMs für jeden Ansatz festgesetzt. Verwandte Arbeiten zeigten in der Vergangenheit aber, dass die Nutzung anderer PLMs die Performance stark verändern kann [11, 21, 24]. Arbeiten sollten deswegen in der Zukunft meine Experimente mit anderen Grundmodellen wiederholen, da so ein vollständigerer Vergleich ermöglicht wird. Dabei können beispielsweise modernere Modelle, wie GPT-5 oder Deepseek-R1, genutzt und/oder *open-source* Modelle verwendet werden.

In dieser Arbeit wurde nur ein FA evaluiert. In der Forschung wurden auch verschiedene andere Ansätze erstellt und evaluiert, die Feinanpassung nutzen, welche in Zukunft zu diesem Vergleich hinzugefügt werden sollte. Diese Ansätze verwenden beispielsweise

LoRa [22], Prompt-Tuning [23], P-Tuning-v2 [27], Dekodierer-PLMs und/oder Lückentextaufgaben. Zusätzlich wenden diese Ansätze teilweise Datenaugmentierung an, wodurch die Trainingsdaten künstlich erweitert werden, ohne dass man mehr annotierte Daten benötigt.

Bei den Experimenten wurde aufgezeigt, dass sowohl RAG als auch *few-shot*/*multi-shot* Prompting verglichen mit *zero-shot* Prompting bessere Ergebnisse erzielen. Zukünftige Arbeiten sollten also überprüfen, ob eine Kombination aus RAG und *few-shot*/*multi-shot* Prompting die Leistung weiter steigern kann. Aufgrund begrenzter Ressourcen war es außerdem nicht möglich, die unterschiedlichen Prompting-Ansätze mit variierenden Prompts zu evaluieren und beim *few-shot*/*multi-shot* Prompting andere TL zu Nicht-TL Verhältnisse zu testen. Zusätzlich konnte nicht getestet werden, wie gut die Leistung von *few-shot*/*multi-shot* Prompting ist, wenn man projekt-externe Beispiele und eine Kombination aus projekt-internen und -externen Beispielen nutzt.

Darüber hinaus konnte in dieser Bachelorarbeit nur die *Requirements Engineering*-Aufgabe TLR betrachtet werden. Wie Feinanpassung und Prompting im Vergleich auf anderen *Requirements Engineering*-Aufgaben, wie beispielsweise der Anforderungsklassifikation, abschneiden, sollte in zukünftigen Arbeiten untersucht werden.

7. Fazit

Das Hauptziel dieser Bachelorarbeit bestand darin, zu ermitteln, mit welcher Anzahl an TBD unter welchen Bedingungen welcher Feinanpassungs- oder Prompting-Ansatz bei der vollautomatisierten TLR von HLRs zu LLRs die beste Performance erzielt. Dazu wurde ein Vergleich auf fünf Datensätzen/Projekten durchgeführt. Für den Vergleich wurden zunächst geeignete TLR-Ansätze aus der Forschung identifiziert und bei Bedarf selbst implementiert. Anschließend wurden vier realistische Szenarien konzipiert, die als Grundlage für den Vergleich dienten und bei denen die beiden TLR-Aufgaben TL-Generierung und TL-Vervollständigung bearbeitet wurden. Insgesamt wurden fünf Experimente durchgeführt, um fehlende Ergebnisse zu ermitteln.

Beim Vergleich zeigte sich, dass *zero-shot* Prompting mit RAG von Fuchß u. a. [11] und Hey u. a. [21] in Bezug auf die F_1-Werte im Durchschnitt der Ansatz mit der besten Performance bei der TL-Generierung ist. Bei dieser müssen alle TLs eines Projekts ermittelt werden. Bei der TL-Vervollständigung, bei der bereits TLs des Projekts vorliegen und der Rest ermittelt werden muss, liefert *zero-shot* Prompting mit RAG bei wenigen vorhandenen TLs die besten Ergebnisse, gemessen an den F_1-Werten. Wenn frühere annotierte Projekte vorliegen, dann übersteigt Feinanpassung *zero-shot* Prompting mit RAG ab ungefähr 20 vorhandenen projekt-internen TLs in den F_1-Werten. Wenn keine früheren annotierten Projekte vorhanden sind, übersteigt Feinanpassung in den F_1-Werten *zero-shot* Prompting mit RAG erst ab etwa 45 vorhandenen TLs.

Die Ergebnisse bedeuten, dass *zero-shot* Prompting mit RAG in Bezug auf die Performance in den meisten Fällen der aktuell beste vollautomatisierte Ansatz für die TLR ist. Wenn bei einem größeren Projekt bereits viele projekt-interne TLs vorliegen und/oder wenn frühere annotierte Projekte vorhanden sind, dann kann sich Feinanpassung möglicherweise lohnen. Bei kleineren Projekten, die eine ähnliche oder geringere Größe wie CM1-NASA oder GANNT haben, ist Feinanpassung im Durchschnitt nicht geeignet, wenn keine früheren annotierten Projekte verfügbar sind. Grund dafür ist, dass für eine bessere Performance mehr TLs benötigt werden, als in den Datensätzen vorhanden sind. Allerdings hängen die Ergebnisse auch stark vom jeweiligen Datensatz ab, was sich insbesondere an Modis zeigt, bei dem Feinanpassung bereits bei deutlich weniger TLs besser wird als Prompting. Damit wird deutlich, dass kein Ansatz universell überlegen ist.

Es ist wichtig hervorzuheben, dass die Validität der Ergebnisse unter Umständen beispielsweise durch die genutzten Testdatensätze beeinträchtigt sein kann. Außerdem ist dieser Vergleich nicht vollständig, da deutlich mehr TLR-Ansätze und -Ansatzvarianten mit unterschiedlichen Variablen (Prompts, Modelle, usw.) existieren, als in dieser Bachelorarbeit

betrachtet werden konnten. Der Vergleich sollte dementsprechend von zukünftigen Arbeiten um neue Ansätze und weitere Testdatensätze erweitert werden.

Literatur

[1] Nouf Alturayeif, Jameleddine Hassine und Irfan Ahmad. „Machine Learning Approaches for Automated Software Traceability: A Systematic Literature Review“. In: *Journal of Systems and Software* 230 (Dez. 2025), S. 112536. ISSN: 0164-1212. DOI: 10.1016/j.jss.2025.112536. (Besucht am 01. 07. 2025).

[2] G. Antoniol u. a. „Recovering Traceability Links between Code and Documentation“. In: *IEEE Transactions on Software Engineering* 28.10 (Okt. 2002), S. 970–983. ISSN: 0098-5589, 1939-3520, 2326-3881. DOI: 10.1109/TSE.2002.1041053. (Besucht am 30. 06. 2025).

[3] Yoshua Bengio u. a. „A Neural Probabilistic Language Model“. In: *Journal of Machine Learning Research 3* (Feb. 2003). (Besucht am 01. 07. 2025).

[4] Tom B. Brown u. a. *Language Models Are Few-Shot Learners*. Juli 2020. DOI: 10.48550/arXiv.2005.14165. arXiv: 2005.14165 [cs]. (Besucht am 15. 07. 2025).

[5] Boqi Chen, Fandi Yi und Dániel Varró. „Prompting or Fine-tuning? A Comparative Study of Large Language Models for Taxonomy Construction“. In: *2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. Västerås, Sweden: IEEE, Okt. 2023, S. 588–596. DOI: 10.1109/models-c59198.2023.00097. (Besucht am 08. 07. 2025).

[6] Davide Dell'Anna, Fatma Başak Aydemir und Fabiano Dalpiaz. „Evaluating Classifiers in SE Research: The ECSER Pipeline and Two Replication Studies“. In: *Empirical Software Engineering* 28.1 (Nov. 2022). ISSN: 1573-7616. DOI: 10.1007/s10664-022-10243-1. (Besucht am 01. 07. 2025).

[7] Yang Deng u. a. „PromptLink: Multi-template Prompt Learning with Adversarial Training for Issue-Commit Link Recovery“. In: *Proceedings of the 18th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. Barcelona Spain: ACM, Okt. 2024, S. 461–467. ISBN: 979-8-4007-1047-6. DOI: 10.1145/3674805.3690751. (Besucht am 03. 06. 2025).

[8] Jacob Devlin u. a. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. Mai 2019. DOI: 10.48550/arXiv.1810.04805. arXiv: 1810.04805 [cs]. (Besucht am 08. 07. 2025).

[9] Romina Etezadi u. a. *Classification or Prompting: A Case Study on Legal Requirements Traceability*. Feb. 2025. DOI: 10.48550/arXiv.2502.04916. arXiv: 2502.04916 [cs]. (Besucht am 03. 06. 2025).

[10] Dominik Fuchß u. a. „Beyond Retrieval: A Study of Using LLM Ensembles for Candidate Filtering in Requirements Traceability“. In: *2025 IEEE 33rd International Requirements Engineering Conference Workshops (REW)*. Valencia, Spain: IEEE, Sep. 2025, S. 5–12. ISBN: 979-8-3315-3834-7. DOI: 10.1109/REW66121.2025.00006. (Besucht am 27. 10. 2025).

[11] Dominik Fuchß u. a. „LiSSA: Toward Generic Traceability Link Recovery through Retrieval-Augmented Generation“. In: *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*. IEEE Computer Society, März 2025, S. 723–723. ISBN: 979-8-3315-0569-1. DOI: 10.1109/ICSE55347.2025.00186. (Besucht am 18. 03. 2025).

[12] Hui Gao u. a. „TRIAD: Automated Traceability Recovery Based on Biterm-enhanced Deduction of Transitive Links among Artifacts“. In: *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. ICSE '24. New York, NY, USA: Association for Computing Machinery, Apr. 2024, S. 1–13. ISBN: 979-8-4007-0217-4. DOI: 10.1145/3597503.3639164. (Besucht am 19. 02. 2025).

[13] Chuyan Ge u. a. „Cross-Level Requirements Tracing Based on Large Language Models“. In: *IEEE Transactions on Software Engineering* (2025), S. 1–23. ISSN: 0098-5589, 1939-3520, 2326-3881. DOI: 10.1109/TSE.2025.3572094. (Besucht am 01. 07. 2025).

[14] Orlena Gotel u. a. „Traceability Fundamentals“. In: *Software and Systems Traceability*. Hrsg. von Jane Cleland-Huang, Orlena Gotel und Andrea Zisman. London: Springer, 2012, S. 3–22. ISBN: 978-1-4471-2239-5. DOI: 10.1007/978-1-4471-2239-5_1. (Besucht am 01. 07. 2025).

[15] Jin Guo, Jinghui Cheng und Jane Cleland-Huang. „Semantically Enhanced Software Traceability Using Deep Learning Techniques“. In: *Proceedings of the 39th International Conference on Software Engineering*. ICSE '17. Piscataway, NJ, USA: IEEE Press, 2017, S. 3–14. ISBN: 978-1-5386-3868-2. DOI: 10.1109/ICSE.2017.9. (Besucht am 01. 07. 2025).

[16] Jin L. C. Guo u. a. „Natural Language Processing for Requirements Traceability“. In: *Handbook on Natural Language Processing for Requirements Engineering*. Hrsg. von Alessio Ferrari und Gouri Ginde. Cham: Springer Nature Switzerland, 2025, S. 89–116. ISBN: 978-3-031-73143-3. DOI: 10.1007/978-3-031-73143-3_4. (Besucht am 13. 05. 2025).

[17] J.H. Hayes, A. Dekhtyar und S.K. Sundaram. „Advancing Candidate Link Generation for Requirements Tracing: The Study of Methods“. In: *IEEE Transactions on Software Engineering* 32.1 (Jan. 2006), S. 4–19. ISSN: 0098-5589. DOI: 10.1109/TSE.2006.3. (Besucht am 30. 06. 2025).

[18] Tobias Hey. „Automatische Wiederherstellung von Nachverfolgbarkeit zwischen Anforderungen und Quelltext“. Diss. 2023. (Besucht am 01. 07. 2025).

[19] Tobias Hey, Jan Keim und Sophie Corallo. „Requirements Classification for Traceability Link Recovery“. In: *2024 IEEE 32nd International Requirements Engineering Conference (RE)*. Reykjavik, Iceland: IEEE, Juni 2024, S. 155–167. ISBN: 979-8-3503-9511-2. DOI: 10.1109/RE59067.2024.00024. (Besucht am 05. 09. 2025).

[20] Tobias Hey u. a. „Improving Traceability Link Recovery Using Fine-grained Requirements-to-Code Relations“. In: *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. Sep. 2021, S. 12–22. DOI: 10.1109/ICSME52107.2021.00008. (Besucht am 01. 07. 2025).

[21] Tobias Hey u. a. „Requirements Traceability Link Recovery via Retrieval-Augmented Generation“. In: *Requirements Engineering: Foundation for Software Quality*. Hrsg. von Anne Hess und Angelo Susi. Cham: Springer Nature Switzerland, 2025, S. 381–397. ISBN: 978-3-031-88531-0. DOI: 10.1007/978-3-031-88531-0_27. (Besucht am 01. 07. 2025).

[22] Edward J. Hu u. a. *LoRA: Low-Rank Adaptation of Large Language Models*. Okt. 2021. DOI: 10.48550/arXiv.2106.09685. arXiv: 2106.09685 [cs]. (Besucht am 02. 07. 2025).

[23] Brian Lester, Rami Al-Rfou und Noah Constant. *The Power of Scale for Parameter-Efficient Prompt Tuning*. Sep. 2021. DOI: 10.48550/arXiv.2104.08691. arXiv: 2104.08691 [cs]. (Besucht am 21. 07. 2025).

[24] Jinfeng Lin u. a. „Enhancing Automated Software Traceability by Transfer Learning from Open-World Data“. In: *CoRR* (Jan. 2022). (Besucht am 01. 07. 2025).

[25] Jinfeng Lin u. a. „Traceability Transformed: Generating More Accurate Links with Pre-Trained BERT Models“. In: *Proceedings of the 43rd International Conference on Software Engineering*. ICSE '21. Madrid, Spain: IEEE Press, Nov. 2021, S. 324–335. ISBN: 978-1-4503-9085-9. DOI: 10.1109/ICSE43902.2021.00040. (Besucht am 01. 07. 2025).

[26] Tianyang Lin u. a. „A Survey of Transformers“. In: *AI Open* 3 (2022), S. 111–132. ISSN: 2666-6510. DOI: 10.1016/j.aiopen.2022.10.001. (Besucht am 18. 07. 2025).

[27] Xiao Liu u. a. *P-Tuning v2: Prompt Tuning Can Be Comparable to Fine-tuning Universally Across Scales and Tasks*. März 2022. DOI: 10.48550/arXiv.2110.07602. arXiv: 2110.07602 [cs]. (Besucht am 21. 07. 2025).

[28] Ali Majidzadeh, Mehrdad Ashtiani und Morteza Zakeri-Nasrabadi. „Multi-Type Requirements Traceability Prediction by Code Data Augmentation and Fine-Tuning MS-CodeBERT“. In: *Computer Standards & Interfaces* 90 (Aug. 2024), S. 103850. ISSN: 09205489. DOI: 10.1016/j.csi.2024.103850. (Besucht am 01. 07. 2025).

[29] Chris Mills, Javier Escobar-Avila und Sonia Haiduc. „Automatic Traceability Maintenance via Machine Learning Classification“. In: *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. Madrid: IEEE, Sep. 2018, S. 369–380. ISBN: 978-1-5386-7870-1. DOI: 10.1109/ICSME.2018.00045. (Besucht am 24. 05. 2025).

[30] Chris Mills u. a. „Tracing with Less Data: Active Learning for Classification-Based Traceability Link Recovery“. In: *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. Sep. 2019, S. 103–113. DOI: 10.1109/ICSME.2019.00020. (Besucht am 01. 07. 2025).

[31] Kevin Moran u. a. „Improving the Effectiveness of Traceability Link Recovery Using Hierarchical Bayesian Networks“. In: *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. ICSE '20. New York, NY, USA: Association for Computing Machinery, Juni 2020, S. 873–885. ISBN: 978-1-4503-7121-6. DOI: 10.1145/3377811.3380418. (Besucht am 01. 07. 2025).

[32] *NASA Systems Engineering Handbook*. (Besucht am 01. 09. 2025).

[33] Feifei Niu u. a. *TVR: Automotive System Requirement Traceability Validation and Recovery Through Retrieval-Augmented Generation*. Apr. 2025. DOI: 10.48550/arXiv.2504.15427. arXiv: 2504.15427 [cs]. (Besucht am 03. 06. 2025).

[34] OpenAI. *GPT-4o-Model*. https://platform.openai.com/docs/models/gpt-4o. (Besucht am 21. 07. 2025).

[35] OpenAI. *OpenAI API Reference — Completions*. https://platform.openai.com/docs/api-reference/completions. (Besucht am 21. 07. 2025).

[36] Branislav Pecher, Ivan Srba und Maria Bielikova. *Comparing Specialised Small and General Large Language Models on Text Classification: 100 Labelled Samples to Achieve Break-Even Performance*. Mai 2025. DOI: 10.48550/arXiv.2402.12819. arXiv: 2402.12819 [cs]. (Besucht am 08. 07. 2025).

[37] Klaus Pohl. *Requirements Engineering: Fundamentals, Principles, and Techniques*. Jan. 2010. ISBN: 978-3-662-51888-5.

[38] Alec Radford u. a. *Improving Language Understanding by Generative Pre-Training*. 2018. (Besucht am 01. 07. 2025).

[39] Alec Radford u. a. *Language Models Are Unsupervised Multitask Learners*. 2019. (Besucht am 01. 07. 2025).

[40] Alberto D. Rodriguez, Katherine R. Dearstyne und Jane Cleland-Huang. „Prompts Matter: Insights and Strategies for Prompt Engineering in Automated Software Traceability“. In: *2023 IEEE 31st International Requirements Engineering Conference Workshops (REW)*. Sep. 2023, S. 455–464. DOI: 10.1109/REW57809.2023.00087. (Besucht am 24. 09. 2024).

[41] Ashish Vaswani u. a. „Attention Is All You Need“. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Red Hook, NY, USA: Curran Associates Inc., 2017, S. 6000–6010. ISBN: 978-1-5108-6096-4.

[42] Joel Walsh u. a. *Fine-Tuning for Better Few Shot Prompting: An Empirical Comparison for Short Answer Grading*. 2025. DOI: 10.48550/ARXIV.2508.04063. (Besucht am 06. 09. 2025).

[43] Bangchao Wang u. a. „MPLinker: Multi-template Prompt-tuning with Adversarial Training for Issue–Commit Link Recovery“. In: *Journal of Systems and Software* 223 (Mai 2025), S. 112351. ISSN: 01641212. DOI: 10.1016/j.jss.2025.112351. (Besucht am 03. 06. 2025).

A. Anhang

A.1. Ergänzende Materialien zu den Datensätzen

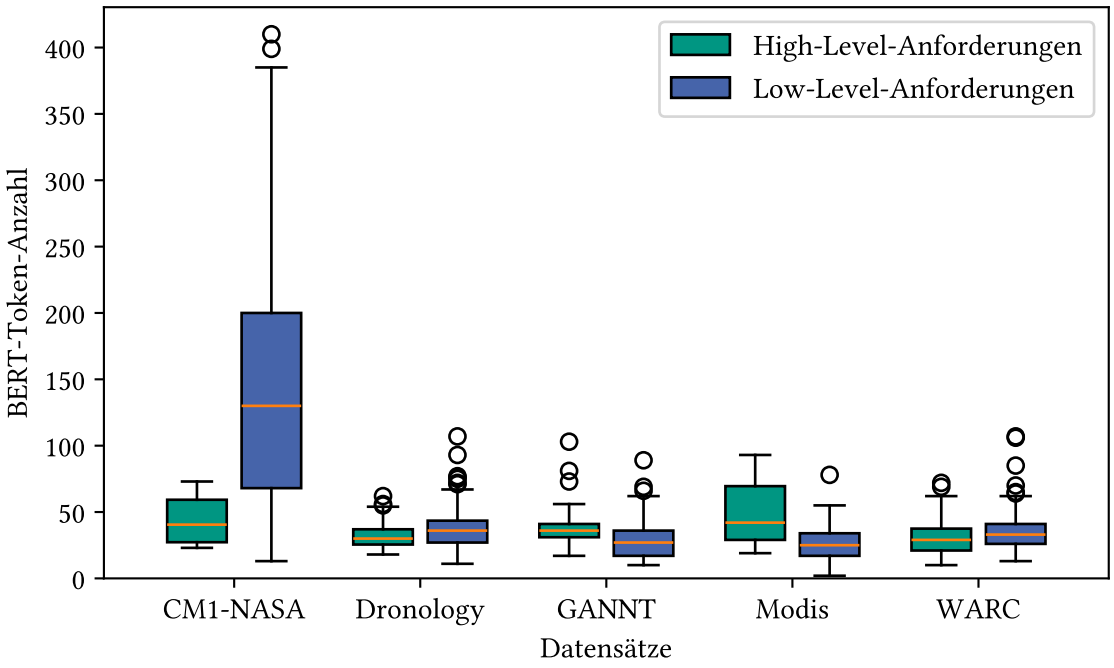

Abbildung A.1.: Anzahl der Tokens für BERT in den Anforderungen der Datensätze

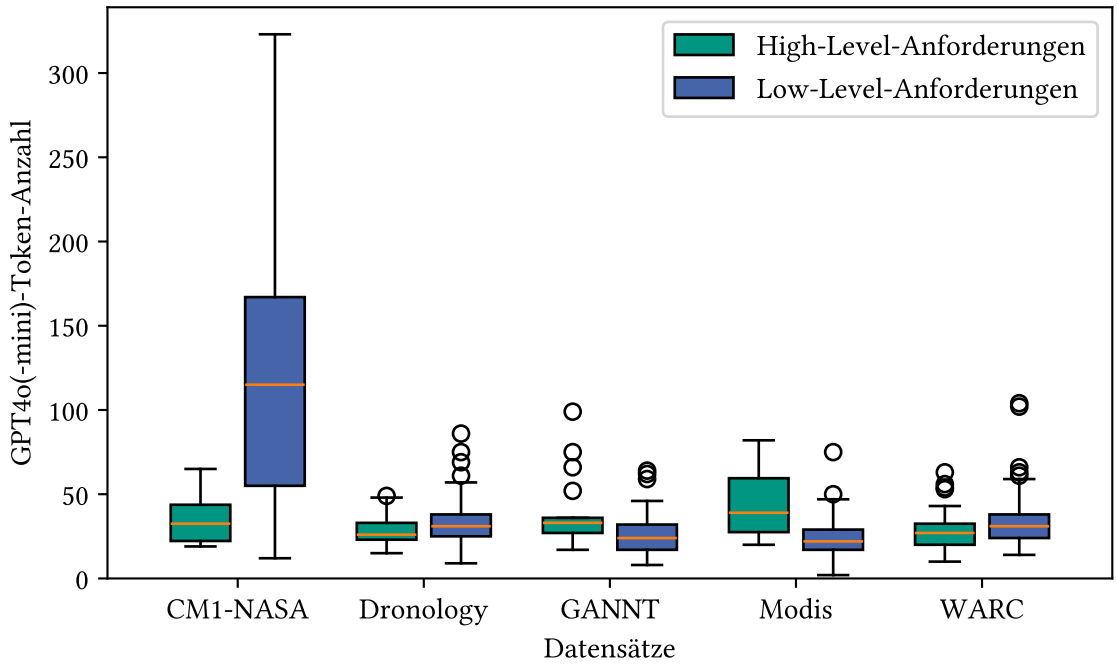

Abbildung A.2.: Anzahl der Tokens für GPT-4o(-mini) in den Anforderungen der Datensätze

A.2. Ergänzende Materialien zu Experiment 3

Trainings-datenanzahl	TL-Anzahl in den Trainings-daten	F_1-Durchschnitt	F_1-Standard-abweichung	F_2-Durchschnitt	F_2-Standard-abweichung
32	1	0,038	0,079	0,031	0,067
64	2	0,088	0,105	0,069	0,084
128	5	0,124	0,121	0,106	0,103
256	10	0,171	0,142	0,164	0,140
512	20	0,334	0,141	0,311	0,138
{932, 933}	36	0,432	0,118	0,414	0,119

Tabelle A.1.: Ergebnisse auf CM1-NASA von Experiment 3 (Feinanpassungsansatz mit *intra-projekt* Datenaufteilung)

Trainings-datenanzahl	TL-Anzahl in den Trainings-daten	F_1-Durchschnitt	F_1-Standard-abweichung	F_2-Durchschnitt	F_2-Standard-abweichung
32	0	0,021	0,000	0,051	0,000
64	1	0,040	0,055	0,047	0,061
128	1	0,033	0,032	0,026	0,027
256	3	0,071	0,053	0,063	0,047
512	5	0,146	0,093	0,126	0,085
1024	11	0,353	0,083	0,313	0,089
2048	22	0,419	0,077	0,397	0,090
4096	43	0,529	0,066	0,525	0,081
8192	86	0,602	0,069	0,614	0,068
16384	173	0,682	0,045	0,694	0,048
{16711, 16712}	176	0,691	0,053	0,700	0,046

Tabelle A.2.: Ergebnisse auf Dronology von Experiment 3 (Feinanpassungsansatz mit *intra-projekt* Datenaufteilung)

Trainings-datenanzahl	TL-Anzahl in den Trainings-daten	F_1-Durchschnitt	F_1-Standard-abweichung	F_2-Durchschnitt	F_2-Standard-abweichung
32	2	0,053	0,088	0,044	0,075
64	4	0,082	0,085	0,076	0,079
128	7	0,147	0,103	0,131	0,103
256	15	0,210	0,108	0,194	0,109
512	{29, 30}	0,287	0,104	0,289	0,123
{938, 939}	{54, 55}	0,343	0,094	0,344	0,113

Tabelle A.3.: Ergebnisse auf GANNT von Experiment 3 (Feinanpassungsansatz mit *intra-projekt* Datenaufteilung)

Trainings-datenanzahl	TL-Anzahl in den Trainings-daten	F_1-Durchschnitt	F_1-Standard-abweichung	F_2-Durchschnitt	F_2-Standard-abweichung
32	1	0,302	0,214	0,343	0,246
64	3	0,415	0,202	0,423	0,233
128	6	0,499	0,211	0,528	0,238
256	11	0,560	0,189	0,570	0,226
512	{22, 23}	0,590	0,132	0,621	0,166
{744, 745}	{32, 33}	0,691	0,110	0,700	0,139

Tabelle A.4.: Ergebnisse auf Modis von Experiment 3 (Feinanpassungsansatz mit *intra-projekt* Datenaufteilung)

Trainings-datenanzahl	TL-Anzahl in den Trainings-daten	F_1-Durchschnitt	F_1-Standard-abweichung	F_2-Durchschnitt	F_2-Standard-abweichung
32	1	0,036	0,042	0,029	0,034
64	2	0,058	0,033	0,051	0,032
128	3	0,089	0,060	0,081	0,060
256	6	0,105	0,071	0,091	0,066
512	12	0,189	0,075	0,159	0,064
1024	25	0,372	0,110	0,346	0,108
2048	{49, 50}	0,530	0,054	0,522	0,064
4096	{99, 100}	0,608	0,082	0,592	0,084
{4485, 4486}	{108, 109}	0,622	0,083	0,621	0,090

Tabelle A.5.: Ergebnisse auf WARC von Experiment 3 (Feinanpassungsansatz mit *intra-projekt* Datenaufteilung)

A.3. Ergänzende Materialien zu Experiment 4

Beispiel-datenanzahl	TL-Anzahl in den Beispieldaten	F_1-Durchschnitt	F_1-Standard-abweichung	F_2-Durchschnitt	F_2-Standard-abweichung
2	1	0,287	0,064	0,473	0,084
4	2	0,316	0,048	0,505	0,070
6	3	0,334	0,070	0,512	0,093
8	4	0,333	0,085	0,510	0,107
10	5	0,319	0,071	0,499	0,099
20	10	0,336	0,063	0,519	0,080
30	15	0,309	0,059	0,496	0,091
40	20	0,327	0,065	0,509	0,081
50	25	0,338	0,090	0,523	0,113
60	30	0,352	0,072	0,536	0,084

Tabelle A.6.: Ergebnisse auf CM1-NASA von Experiment 4 (*few-shot*/*multi-shot* Prompting (PA3) mit *intra-projekt* Datenaufteilung)

Beispiel-datenanzahl	TL-Anzahl in den Beispieldaten	F_1-Durchschnitt	F_1-Standard-abweichung	F_2-Durchschnitt	F_2-Standard-abweichung
2	1	0.257	0.040	0.439	0.048
4	2	0.259	0.069	0.434	0.077
6	3	0.287	0.051	0.463	0.049
8	4	0.270	0.029	0.457	0.032
10	5	0.298	0.078	0.474	0.071
20	10	0.280	0.043	0.465	0.042
30	15	0.330	0.039	0.509	0.039
40	20	0.341	0.032	0.506	0.033
50	25	0.351	0.036	0.524	0.028
60	30	0.353	0.045	0.516	0.038

Tabelle A.7.: Ergebnisse auf Dronology von Experiment 4 (*few-shot*/*multi-shot* Prompting (PA3) mit *intra-projekt* Datenaufteilung)

Beispiel-datenanzahl	TL-Anzahl in den Beispieldaten	F_1-Durchschnitt	F_1-Standard-abweichung	F_2-Durchschnitt	F_2-Standard-abweichung
2	1	0,379	0,049	0,578	0,050
4	2	0,400	0,054	0,591	0,058
6	3	0,413	0,060	0,601	0,063
8	4	0,419	0,046	0,607	0,056
10	5	0,428	0,037	0,609	0,058
20	10	0,433	0,043	0,592	0,056
30	15	0,451	0,045	0,585	0,067
40	20	0,473	0,054	0,621	0,055
50	25	0,498	0,054	0,621	0,061
60	30	0,493	0,061	0,621	0,069

Tabelle A.8.: Ergebnisse auf GANNT von Experiment 4 (*few-shot*/*multi-shot* Prompting (PA3) mit *intra-projekt* Datenaufteilung)

Beispiel-datenanzahl	TL-Anzahl in den Beispieldaten	F_1-Durchschnitt	F_1-Standard-abweichung	F_2-Durchschnitt	F_2-Standard-abweichung
2	1	0,498	0,092	0,549	0,114
4	2	0,527	0,088	0,604	0,137
6	3	0,543	0,097	0,620	0,147
8	4	0,567	0,099	0,644	0,128
10	5	0,553	0,149	0,626	0,175
20	10	0,559	0,125	0,596	0,151
30	15	0,551	0,135	0,587	0,163
40	20	0,589	0,102	0,639	0,124
50	25	0,569	0,098	0,630	0,126
60	30	0,559	0,095	0,636	0,128

Tabelle A.9.: Ergebnisse auf Modis von Experiment 4 (*few-shot*/*multi-shot* Prompting (PA3) mit *intra-projekt* Datenaufteilung)

Beispiel-datenanzahl	TL-Anzahl in den Beispieldaten	F_1-Durchschnitt	F_1-Standard-abweichung	F_2-Durchschnitt	F_2-Standard-abweichung
2	1	0.375	0.090	0.542	0.082
4	2	0.400	0.073	0.575	0.069
6	3	0.423	0.060	0.592	0.062
8	4	0.438	0.067	0.607	0.070
10	5	0.461	0.063	0.628	0.075
20	10	0.492	0.077	0.637	0.082
30	15	0.483	0.065	0.642	0.072
40	20	0.492	0.071	0.646	0.087
50	25	0.482	0.064	0.639	0.076
60	30	0.511	0.059	0.658	0.071

Tabelle A.10.: Ergebnisse auf WARC von Experiment 4 (*few-shot*/*multi-shot* Prompting (PA3) mit *intra-projekt* Datenaufteilung)

A.4. Ergänzende Materialien zu Experiment 5

projekt-interne Trainings-datenanzahl	TL-Anzahl in den projekt-internen Trainings-daten	F_1-Durchschnitt	F_1-Standard-abweichung	F_2-Durchschnitt	F_2-Standard-abweichung
32	1	0,267	0,170	0,222	0,148
64	2	0,288	0,138	0,235	0,118
128	5	0,298	0,212	0,254	0,195
256	10	0,371	0,196	0,324	0,170
512	20	0,443	0,086	0,384	0,070
{932, 933}	36	0,459	0,095	0,398	0,064

Tabelle A.11.: Ergebnisse auf CM1-NASA von Experiment 5 (Feinanpassungsansatz mit *intra-cross-projekt* Datenaufteilung)

projekt-interne Trainings-datenanzahl	TL-Anzahl in den projekt-internen Trainings-daten	F_1-Durchschnitt	F_1-Standard-abweichung	F_2-Durchschnitt	F_2-Standard-abweichung
32	0	0,459	0,052	0,551	0,049
64	1	0,464	0,045	0,564	0,041
128	1	0,490	0,038	0,530	0,023
256	3	0,523	0,048	0,506	0,053
512	5	0,539	0,031	0,525	0,038
1024	11	0,500	0,064	0,450	0,079
2048	22	0,535	0,057	0,482	0,054
4096	43	0,579	0,046	0,539	0,065
8192	86	0,652	0,062	0,629	0,078
16384	173	0,714	0,047	0,698	0,046
{16711, 16712}	176	0,710	0,039	0,691	0,061

Tabelle A.12.: Ergebnisse auf Dronology von Experiment 5 (Feinanpassungsansatz mit *intra-cross-projekt* Datenaufteilung)

projekt-interne Trainings-datenanzahl	TL-Anzahl in den projekt-internen Trainings-daten	F_1-Durchschnitt	F_1-Standard-abweichung	F_2-Durchschnitt	F_2-Standard-abweichung
32	2	0,357	0,152	0,316	0,143
64	4	0,285	0,124	0,253	0,115
128	7	0,321	0,132	0,285	0,152
256	15	0,328	0,118	0,303	0,117
512	{29, 30}	0,350	0,125	0,334	0,135
{938, 939}	{54, 55}	0,436	0,092	0,442	0,118

Tabelle A.13.: Ergebnisse auf GANNT von Experiment 5 (Feinanpassungsansatz mit *intra-cross-projekt* Datenaufteilung)

projekt-interne Trainings-datenanzahl	TL-Anzahl in den projekt-internen Trainings-daten	F_1-Durchschnitt	F_1-Standard-abweichung	F_2-Durchschnitt	F_2-Standard-abweichung
32	1	0,287	0,197	0,223	0,160
64	3	0,582	0,182	0,499	0,198
128	6	0,617	0,172	0,557	0,193
256	11	0,646	0,137	0,594	0,175
512	{22, 23}	0,708	0,117	0,691	0,139
{744, 745}	{32, 33}	0,783	0,098	0,775	0,121

Tabelle A.14.: Ergebnisse auf Modis von Experiment 5 (Feinanpassungsansatz mit *intra-cross-projekt* Datenaufteilung)

projekt-interne Trainings-datenanzahl	TL-Anzahl in den projekt-internen Trainings-daten	F_1-Durchschnitt	F_1-Standard-abweichung	F_2-Durchschnitt	F_2-Standard-abweichung
32	1	0,370	0,130	0,352	0,144
64	2	0,356	0,169	0,302	0,153
128	3	0,386	0,149	0,324	0,125
256	6	0,420	0,142	0,366	0,136
512	12	0,455	0,097	0,410	0,113
1024	25	0,505	0,053	0,459	0,063
2048	{49, 50}	0,565	0,086	0,526	0,088
4096	{99, 100}	0,652	0,099	0,630	0,103
{4485, 4486}	{108, 109}	0,631	0,078	0,615	0,091

Tabelle A.15.: Ergebnisse auf WARC von Experiment 5 (Feinanpassungsansatz mit *intra-cross-projekt* Datenaufteilung)

A.5. Ergänzende Materialien zu allen Szenarien

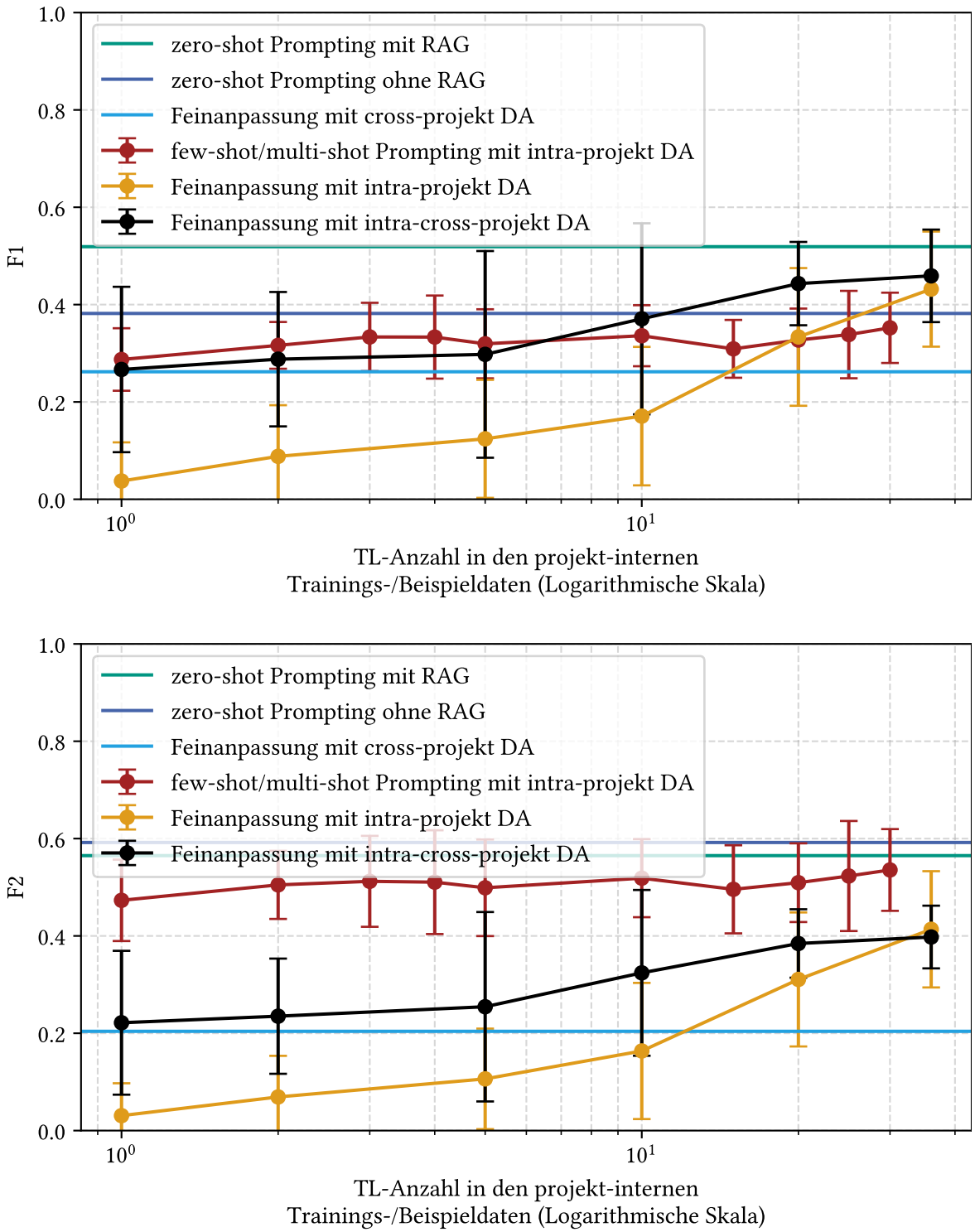

Abbildung A.3.: Ergebnisse aller Ansätze auf CM1-NASA - Durchschnittliche F_1-Werte mit Standardabweichungen bzw. F_1-Werte (oben) und durchschnittliche F_2-Werte mit Standardabweichungen bzw. F_2-Werte (unten)

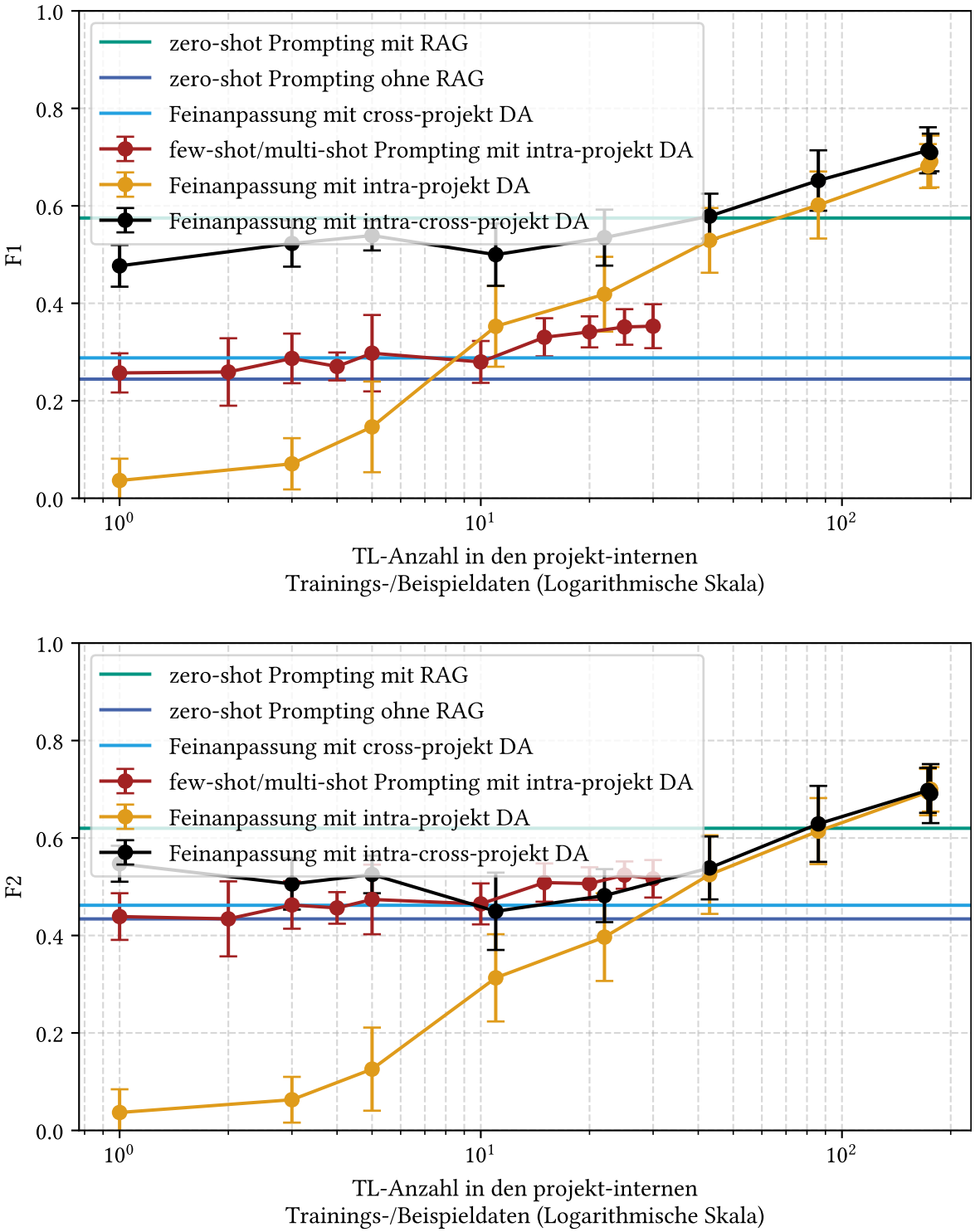

Abbildung A.4.: Ergebnisse aller Ansätze auf Dronology - Durchschnittliche F_1-Werte mit Standardabweichungen bzw. F_1-Werte (oben) und durchschnittliche F_2-Werte mit Standardabweichungen bzw. F_2-Werte (unten)

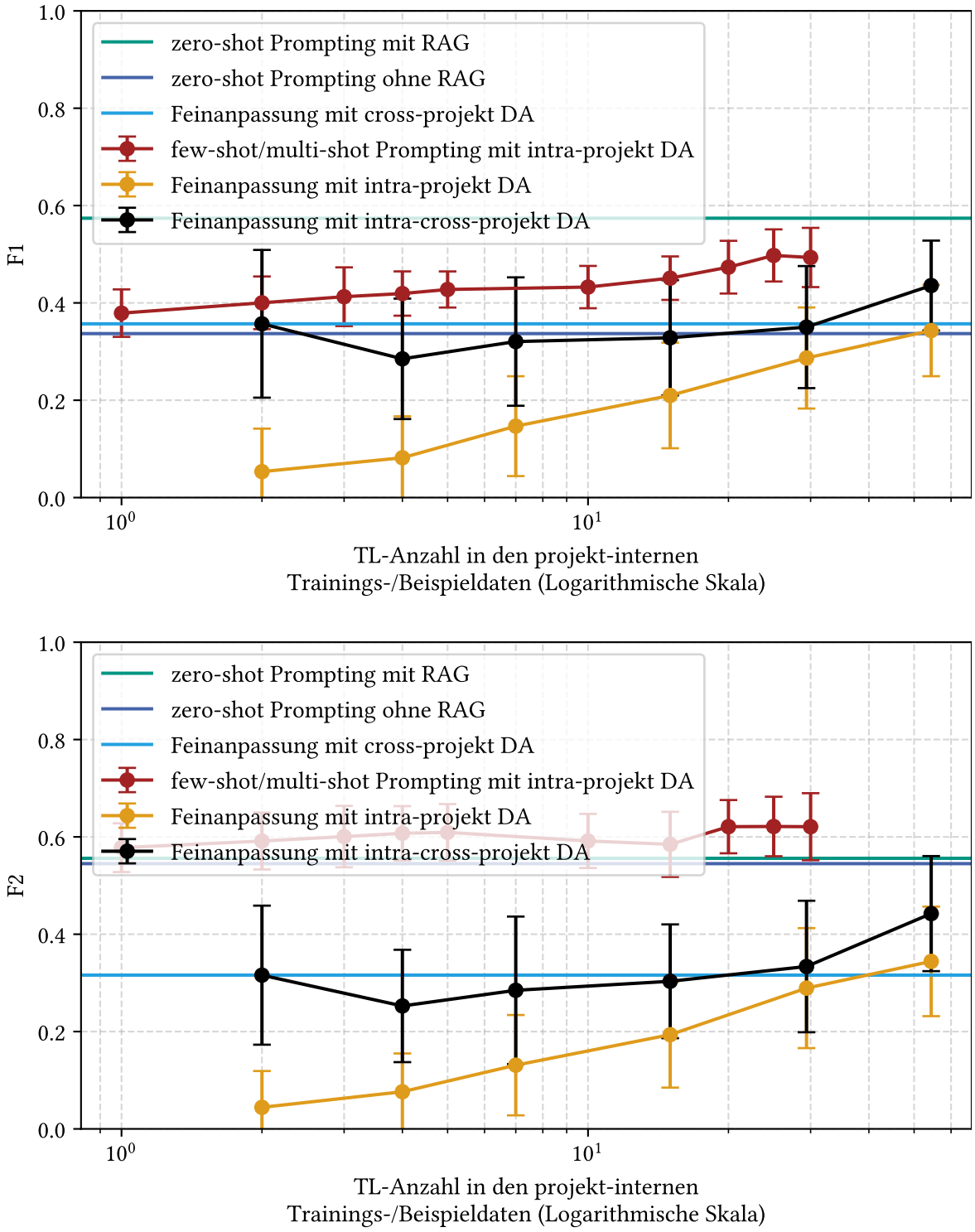

Abbildung A.5.: Ergebnisse aller Ansätze auf GANNT - Durchschnittliche F_1-Werte mit Standardabweichungen bzw. F_1-Werte (oben) und durchschnittliche F_2-Werte mit Standardabweichungen bzw. F_2-Werte (unten)

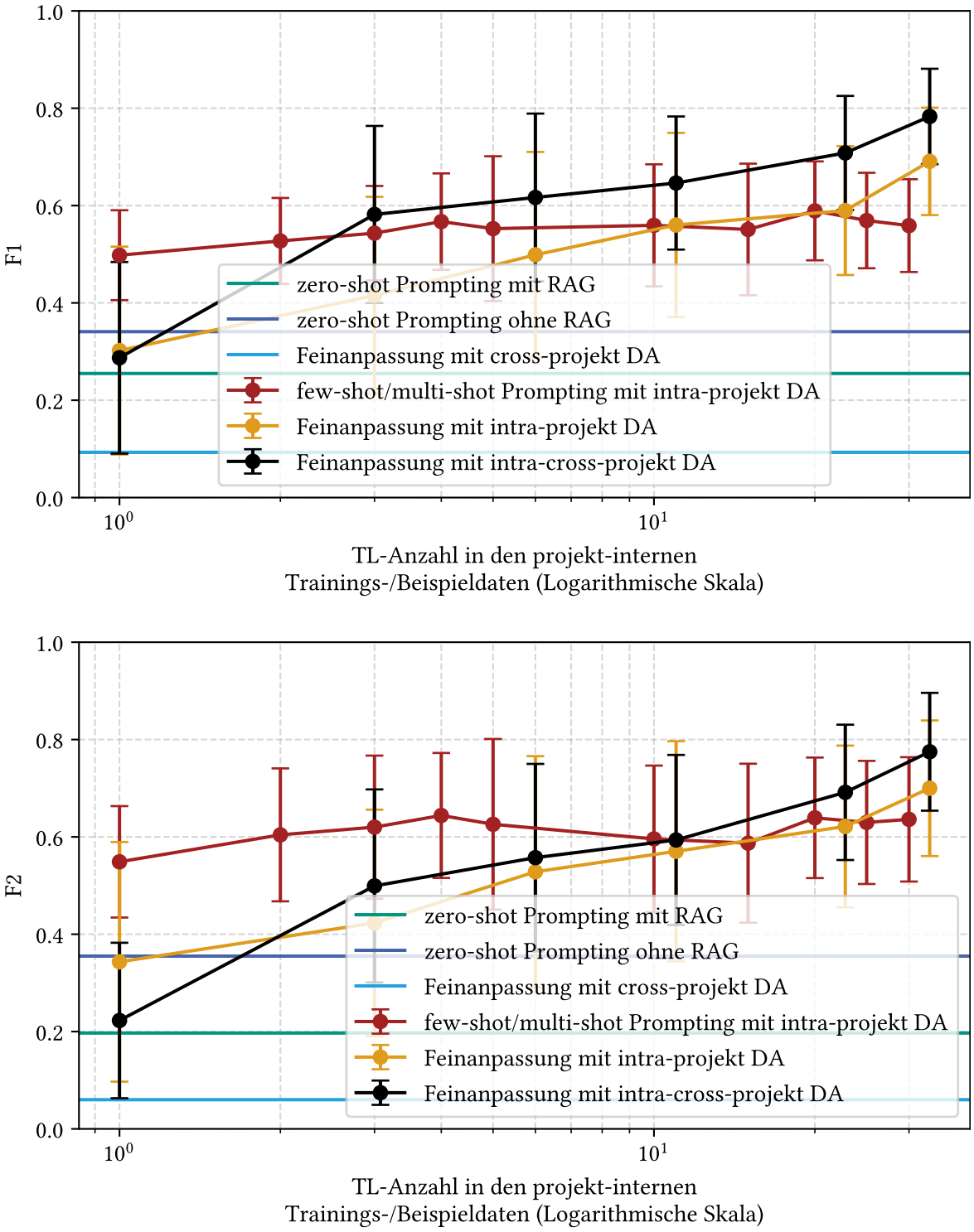

Abbildung A.6.: Ergebnisse aller Ansätze auf Modis - Durchschnittliche F_1-Werte mit Standardabweichungen bzw. F_1-Werte (oben) und durchschnittliche F_2-Werte mit Standardabweichungen bzw. F_2-Werte (unten)

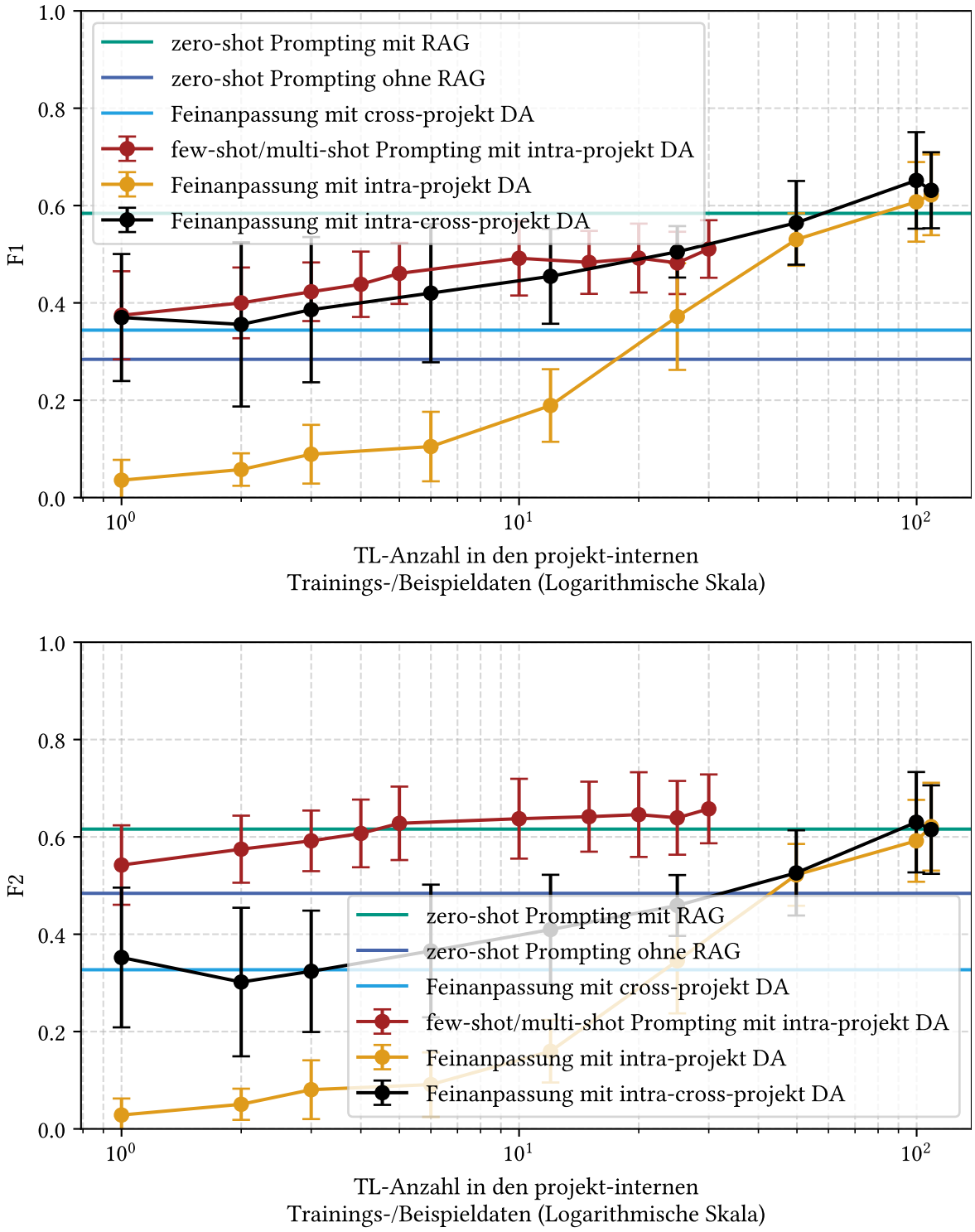

Abbildung A.7.: Ergebnisse aller Ansätze auf WARC - Durchschnittliche F_1-Werte mit Standardabweichungen bzw. F_1-Werte (oben) und durchschnittliche F_2-Werte mit Standardabweichungen bzw. F_2-Werte (unten)