Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

Deutsches Zentrum
für Luft- und Raumfahrt

Bachelorarbeit

Bachelor of Science (B.Sc.) Wirtschaftsinformatik

Semi-automatische Annotation von Webdaten mit Georeferenzierung

von Yunus Emre Sirin

9040908

Fachbereich Informatik

Erstbetreuer	**Prof. Dr. Andreas Hackelöer**
Zweitbetreuer	**Prof. Dr. Sascha Alda**
Externer Betreuer	**Dr. Tobias Hecking**

Eingereicht am: **16. Dezember 2024**

Eidesstattliche Erklärung

Ich versichere hiermit, die von mir vorgelegte Arbeit selbstständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen. Mir ist bewusst, dass sich die Hochschule vorbehält, meine Arbeit auf plagiierte Inhalte hin zu überprüfen und dass das Auffinden von plagiierten Inhalten zur Nichtigkeit der Arbeit führen kann.

Ort, Datum	Unterschrift

Danksagung

An dieser Stelle möchte ich all jenen danken, die mich während meines Studiums und der Anfertigung dieser Arbeit begleitet haben. Insbesondere möchte ich meinen Dank an meine Betreuer, dem Herrn. Prof. Dr. Andreas Hackelöer und dem Herrn Prof. Dr. Sascha Alda, und an meinen externen Betreuer bei dem Deutschen Zentrum für Luft- und Raumfahrt, dem Herrn Dr. Tobias Hecking, aussprechen, die mich durch ihre Beratung, Betreuung und Ressourcenbereitstellung unterstützt und die Bewältigung von unvorhergesehenen Hürden erleichtert haben. Außerdem danke ich meiner Familie und meinen Freunden, die mich bei der stressigen Anfertigung der Arbeit motiviert haben.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

1.1 Hintergrund und Motivation

Geographische Informationen aus großen Mengen unstrukturierter Webdaten zu extrahieren ist für zahlreiche Anwendungen von großer Bedeutung, wie etwa in der Standortplanung, in der Umweltforschung oder bei dem Katastrophenmanagement [1]. Daher ist durch die Entwicklung einer Methode zur Automatisierung dieser Prozesse und einer besseren Extraktion von Georeferenzen eine signifikante Effizienzsteigerung zu erwarten.

Die Georeferenzierung von Webdaten ist ein wichtiger zentraler Aspekt in der geographischen Informationsverarbeitung [1] und ermöglicht eine Vielzahl von Anwendungen wie die lokale Informationssuche, die Unterstützung im Krisenmanagement, geographische Empfehlungssysteme, etc. Darunter versteht man die Zuweisung von geographischen Koordinaten, die Georeferenz, zu einem oder mehreren Datensätzen, die in verschiedenen Anwendungen wie in Geoinformationssystemen oder in der Computerkartografie genutzt werden. Aktuelle Methoden zur Georeferenzierung basieren auf regelbasierten Ansätzen sowie traditionellen maschinellen Lernmodellen. Mit dem Aufkommen von neuen leistungsstarken Large Language Models (Englisch für „Großes Sprachmodell“ und kurz: LLMs) ergeben sich weitere neue Möglichkeit, aus unstrukturierten Texten Georeferenzen automatisiert zu extrahieren und die Qualität der Georeferenzierung zu verbessern.

Der praktische Bezug zur Geoinformatik in meinem Praktikum bei dem Deutschen Zentrum für Luft- und Raumfahrt (kurz: DLR) in der Abteilung „Intelligente und verteilte Softwaresysteme“ am Institut für Softwaretechnologie hat mich dazu veranlasst, auf Absprache mit meinem externen Betreuer, dem Herrn Dr. Tobias Hecking, dieses Thema zu wählen. Das Thema meiner Bachelorarbeit hat unter anderem seinen Ursprung im europäischen Projekt OpenWebSearch.EU [1] [2] [2], das sich zum Ziel gesetzt hat, Europas Unabhängigkeit im Suchmaschinenmarkt zu fördern, in dem ein offenes Web-Index (auf Englisch „Open Web Index“ und kurz: OWI) geschaffen und eine auf europäischen Werten basierende Web-Such- und Analyse-Infrastruktur (auf Englisch „Open Web Search and Analysis Infrastructure“ und kurz: OWSAI) entwickelt wird. Diese Infrastruktur soll eine demokratische, transparente und innovative Alternative zu den bestehenden marktbeherrschenden Anbietern wie Google oder Microsoft sein. Dadurch wird in Europa die Möglichkeit geboten werden, das Web systematisch als Ressource zu nutzen, insbesondere für KI-Innovationen und Forschungsprojekte, ohne auf externe Anbieter angewiesen zu sein. Eine der zentralen Herausforderungen bei

[1] https://openwebsearch.eu

[2] https://www.dlr.de/de/sc/forschung-transfer/projekte/openwebsearch.eu

der Umsetzung des Projektes besteht darin, geographische Informationen, die derzeit in großem Maßstab nicht verfügbar sind, aus Webdaten zu extrahieren [1, vgl. Kapitel 3] [3, vgl. Kapitel 1] [4, vgl. Kapitel 4]. Daher hatte ich Rahmen meines Praxisprojektes die Aufgabe, ein Annotationstool zu entwickeln, mit dem geographische Daten auf verknüpften Text- und Kartenansichten angezeigt sowie Georeferenzen manuell annotiert und verwaltet werden können. Um die Effizienz bei der Annotation größerer Datenmengen zu steigern, soll dieses Werkzeug um ein großes Sprachmodell (auf Englisch „Large Language Model“ und kurz: LLM) erweitert werden, welches Daten automatisiert vorannotiert. Diese initialen Annotationen können bei Bedarf manuell korrigiert werden, wodurch sowohl die Annotationen verbessert werden als auch das LLM nachtrainiert werden kann. Die Herausforderung, ein Annotationstool mit einem integrierten LLM zu entwickeln, fasziniert mich. Obwohl LLMs in vielen Bereichen der Sprachverarbeitung bedeutende Fortschritte erzielt haben, ist ihre Anwendung in der (semi-)automatischen Georeferenzierung bisher nur vereinzelt untersucht worden [5, 6, 7].

1.2 Zielsetzung und Fragestellung

Das Hauptziel der Arbeit ist es, der zentralen Fragestellung nachzugehen, die wie folgt lautet: Wie können LLMs effektiv zur semi-automatischen Annotation von Webdaten mit Georeferenzen in dem eigens entwickelten Annotationstool eingesetzt und trainiert werden? Daraus erschließen sich die weiteren Fragen, welche LLMs am besten für die Extraktion von Georeferenzen geeignet sind, wie ein benutzerfreundliches Interface zur Korrektur und Verfeinerung der automatisch extrahierten Georeferenzen gestaltet werden kann und wie die Integration des Frameworks MLflow zur Verwaltung und Verfolgung von Trainingsprozessen und Experimenten erfolgen könnte.

Bisher erfolgt die Annotation von Webdaten mit Georeferenzierung oft durch regelbasierte Systeme oder konventionelle maschinelle Lernmodelle, die in der Regel auf explizit gekennzeichnete Trainingsdaten angewiesen sind [8, vgl. Kapitel 3.1]. Diese Methoden erfordern größtenteils manuelle Eingriffe, sowohl bei der Datenaufbereitung als auch bei der Nachbearbeitung der Ergebnisse. Aufgrund der Regelbasiertheit dieser Systeme sind sie nicht flexibel genug, um mit der Vielfalt und Komplexität unstrukturierter Webdaten umzugehen [1, 9]. Eine geringe Abdeckung unbekannter Ortsnamen und die fehlerhafte Georeferenzierung sind häufig Herausforderungen, die die Effizienz und Genauigkeit der bestehenden Methoden einschränken [4, vgl. Kapitel 2.2].

LLMs haben die Fähigkeit, semantisch komplexe Zusammenhänge in Texten zu erkennen, und können so Georeferenzen auch in unstrukturierten Daten zuverlässig identifizieren und extrahieren [10]. Diese Modelle werden auf großen allgemeinen als auch spezifischen Daten trainiert und sind dadurch in der Lage, auch seltene Ortsnamen oder unübliche Schreibweisen zu erkennen, was eine wesentliche Leistungsverbesserung gegenüber den klassischen Ansätzen darstellt [11, 12, 13].

Die Arbeitshypothese lautet, dass der Einsatz von LLMs in dem eigens entwickelten Annotationstool die Präzision der Georeferenzierung steigern und die iterative Verbesserung dieser LLMs die Qualität der Georeferenzierung verbessern kann, da sie wegen der reduzierten Abhängigkeit von explizit gekennzeichneten Daten auch kontextuelle Informationen besser nutzen können, was den manuellen Nachbearbeitungsaufwand erheblich reduzieren würde. Dadurch würden Trainingsdaten prinzipiell effektiver erstellt werden können.

Kapitel 2

Grundlagen

2.1 Georeferenzierung

Die Georeferenzierung ist der Prozess der Zuordnung von geographischen Objekten zu einem geographischen Koordinatensystem, also die Verknüpfung von räumlichen Informationen mit geographischen Koordinaten [14]. Geographische Objekte lassen sich in in drei verschiedene Informationstypen unterteilen:

- Geometrische Informationen beschreiben geometrische Eigenschaften eines Objektes, wie z.B. den Verlauf oder die Form eines Straßenabschnitts.
- Topologische Informationen beziehen sich auf die Eigenschaft geographischer Objekte, die unverändert bleiben, wenn die Objekte verformt werden (z.B. Anzahl der Straßen an einer Kreuzung). Im Kontekt der Georeferenzierung ist die Struktur von Netzwerken wie Straßen besonders relevant, da sie die Beziehung zwischen diesen Objekten beschreibt.
- Semantische Informationen umfassen die inhaltlichen Eigenschaften, die einem geographischen Ort zugeordnet werden können, wie z.B. die Orts- oder Straßennamen. Methoden der Georeferenzierung nutzen diese Eigenschaften, allein oder in Kombination mit anderen Informationstypen, um geographische Objekte eindeutig zu identifizieren.

Ziel der Georeferenzierung ist die Integration von verschiedenen räumlichen Daten in ein kohärentes, einheitliches Koordinatensystem. Dies ist besonders wichtig für geographische Informationssysteme (kurz: GIS), die geographische Informationen analysieren oder darstellen möchten. Traditionell erfolgt die Georeferenzierung durch manuelle Annotationen oder durch die Verwendung von speziell angefertigten Softwaretools, die geographische Datenbanken durchsuchen. Jedoch ist die manuelle Annotation sehr mühselig, da die Daten einzeln abgeglichen, die räumlichen Informationen erschlossen und kontextualisiert werden müssen. Anschließend erfolgt die Zuweisung der geographischen Koordinaten (wie Breiten- und Längengrade) zu den identifizierten räumlichen Entitäten.

2.2 Large Language Models

Unter einem Large Language Model versteht man ein leistungsstarkes Modell, das in der Lage ist, menschliche Sprache zu verstehen und zu generieren. Diese Modelle können textuelle Inhalte in ihren Kernaspekten erschließen und Informationen extrahieren [15, 16]. Diese Fähigkeit ist zurückzuführen auf das tiefe neuronale

Netzwerk (auf Englisch „Deep Neural Network“), das große Textdatenmengen analysiert, um Sprachmuster zu erkennen. Neuronale Netzwerke bestehen aus verbundenen, in Schichten organisierten Knoten, sogenannten Neuronen, die es dem Modell ermöglichen, Sprachzusammenhänge zu erkennen [17]. Durch diese Architektur können LLMs sprachbezogene Aufgaben lösen und kohärente Antworten liefern. Aufgrund dieser besonderen Charakteristika des LLMs weist die Nutzung ein erhebliches Potential auf, um Georeferenzierungen zu automatisieren. Neue große Sprachmodelle haben daher an Bedeutung gewonnen, die zu diesem Zweck, in einigen Fällen sogar ohne Training oder eine Feinabstimmung, angewendet werden können. Im Folgenden fokussieren wir uns auf die Modelle der drei wesentlichen Hauptakteure Google, OpenAI und Meta (ehemals Facebook), die an der Entwicklung von leistungsstarken Sprachmodellen beteiligt sind.

BERT

BERT (Englisch für „Bidirectional Encoder Representations from Transformers“ und auf Deutsch „Bidirektionale Encoder-Darstellungen aus Transformatoren“), entwickelt von Google und beschrieben von Devlin et al. [11], stellt einen bedeutenden Fortschritt in der natürlichen Sprachverarbeitung (auf Englisch „Natural language processing“ und kurz: NLP) dar. Das Sprachmodell basiert auf einer sogenannte Transformer-Architektur. Dabei handelt es sich um eine Architektur, die es erlaubt, nach dem Mechanismus „Self-Attention“ (auf Deutsch: Selbstaufmerksamkeit) zu arbeiten [18]. Das bedeutet, dass die Bedeutungen und semantischen Beziehungen zwischen allen Wörtern in einem Satz unabhängig von ihrer Position parallel, also bidirektional [1] erfasst werden. Dies ermöglicht es, kontextuelle Zusammenhänge über lange syntaktische Distanzen hinweg zu erkennen und zu verstehen, während gleichzeitig die Effizienz der Sprachverarbeitung gesteigert wird. BERT bildet in der Praxis weitestgehend das Grundfundament für große Sprachmodelle und wird beispielsweise für die Klassifizierung von Text oder die Named Entity Recognition (kurz: NER und auf Deutsch „Erkennung benannter Entitäten“), also das Erkennen von klassifizierten Entitäten wie Personen oder Orte, verwendet [19], was es zu einem wünschenswerten Tool für die semantische Analyse bei der semi-automatischen Annotation macht.

GPT

GPT (Englisch für „Generative Pretrained Tranformer“ und auf Deutsch „Generativ vortrainierter Transformator“), beschrieben von Brown et al. [12], ist ein leistungsstarkes state-of-the-art [2] Sprachmodell, das darauf trainiert ist, menschenähnliche Texte zu generieren. Ähnlich wie BERT basiert dieses Modell auf der Transformer-Architektur und wurde in mehreren Versionen (z.B. GPT-2, GPT-3, GPT-4) von OpenAI veröffentlicht. GPT kann effektiv konsistente und gut-kontextualisierte Texte erzeugen, Fragen beantworten, Zusammenfassungen aller Längen erstellen und jene Aufgaben bewältigen, worauf es feinjustiert ist. Die Besonderheit von GPT liegt darin, dass es aufgrund seines Trainingskonzeptes sowohl allgemeine als auch spezifische Kenntnisse in seinen Antworten integriert. Daher wird es für verschiedene Zwecke eingesetzt, angefangen von der automatisierten Textgenerierung bis hin zur semantischen Analyse in komplexen Systemen. Dies ist besonders relevant für die Georeferenzierung, um geographische Informationen in umfangreichen, semantisch komplexen Texten abzuleiten. Das Modell kann zudem per API-Aufruf oder modular verwendet werden. Eine API (kurz für „Application Programming Interface“) ist eine Schnittstelle, die den Austausch von Funktionen und Daten zwischen verschiedenen Softwareanwendungen ermöglicht, ohne dass der Nutzer die interne Arbeitsweise verstehen muss.

[1] Die Wörter werden gleichzeitig von links und rechts analysiert
[2] Technologisch auf dem neuesten Stand

LLaMA

LLaMA (kurz für „Large Language Model Meta AI“) ist ein Sprachmodell, das von Meta [13] entwickelt wurde und von Anfang an speziell für die Forschung und Entwicklung in Bezug auf künstliche Intelligenz konzipiert wurde. Ähnlich wie die anderen Sprachmodelle basiert auch LLaMA auf der Transformer-Architektur und wird für das Verstehen und Generieren von natürlicher Sprache eingesetzt. Es wurde in verschiedenen Versionen veröffentlicht (wie LLaMA 1, LLaMA 2 etc.), die jeweils durch umfangreiches Training verbessert wurden. Im Gegensatz zu Modellen wie GPT ist LLaMA ressourcenschonender und erzielt daher ähnliche oder sogar bessere Ergebnisse bei geringerer Rechenleistung, was es besonders attraktiv für Forschungsprojekte mit begrenzten Ressourcen macht. LLaMA kann Aufgaben wie Textgenerierung, Textverständnis und allgemeinere Sprachverarbeitung übernehmen. Anders als GPT gibt es zwar keine API-Integration, jedoch wird LLaMA als Open-Source beworben und kann von Entwicklern und Forschern zu eigenen Zwecken genutzt und implementiert werden. Das DLR nutzt die Technologie auf selbst maßgeschneiderten Hostseiten und mit angepasster API-Integration für diverse Forschungszwecke.

2.3 GeoSense Annotator

Der GeoSense Annotator ist im Rahmen meines Praxisprojektes beim DLR entstanden und ist ein web-basiertes Content-Management-Tool zur manuellen Annotation und Verwaltung von (Web-)Daten mit Georeferenzen. Die Idee bei dem Projekt bestand darin, fehlende oder fehlerbehaftete Georeferenzen, die mit Geoparsern [3] nach Ortsnamen und den zugehörigen Koordinaten geprüft wurden, zu ergänzen und zu korrigieren. Entwickelt wurde die Software mit React im Frontend und FastAPI im Backend. React [4] ist eine Open-Source-JavaScript-Bibliothek, die zur Entwicklung von Benutzeroberflächen verwendet wird. Sie wird hauptsächlich für die Erstellung von Single-Page-Applications (Englisch für „Einzelseitenanwendungen“ und kurz: SPAs) genutzt, bei denen die Inhalte dynamisch aktualisiert werden, ohne die gesamte Seite neu zu laden. Unter anderem wird für das Erstellen attraktiver und benutzerfreundlicher Oberflächen die UI-Komponentenbibliothek Material UI [5] verwendet, die vorgefertigte Komponenten wie Buttons, Karten und Dialogfenster anbietet. FastAPI[6] ist ein leistungsstarkes Web-Framework zur Erstellung von APIs in Python. Es wurde für hohe Leistung und Benutzerfreundlichkeit entwickelt und unterstützt Funktionen wie asynchrone Programmierung, automatische Dokumentation und Typprüfung.

Das Frontend umfasst eine Startseite mit einer Navigationsleiste und einer rasterförmigen Benutzeroberfläche. Auf dem vier-flächigen Raster sind die einzelnen React-Komponenten dargestellt, die selber in mehrere einzelne Komponenten unterteilt sind (siehe Abbildung 2.1). Auf der linken Seite des Prototyps befindet sich der FileExplorers. Diese Komponente dient dazu, über eine Upload-Komponente beliebig viele Dokumente hochzuladen, welche dann als Liste von Einträgen abgebildet werden. Hierbei werden Daten wie der Dateiname, der Bearbeitungszustand (in Form eines grünen Symbols) und ein Menü-Button angezeigt. Über den Menü-Button bei den einzelnen Dateien können Änderungen vorgenommen werden, wie zum Beispiel das Löschen oder das Herunterladen einer Datei. Wird eine Datei bearbeitet, erscheint das grüne Symbol in Rot. Es besteht unter Anderem auch die Möglichkeit, alle Dateien, inklusive Änderungen, abzuspeichern (über den „Save all files“-Button auf der unteren Leiste des FileExplorers).

[3] vgl. Kapitel 2.4.1 Geoparser
[4] https://reactjs.org/
[5] https://mui.com/
[6] `https://fastapi.tiangolo.com/`

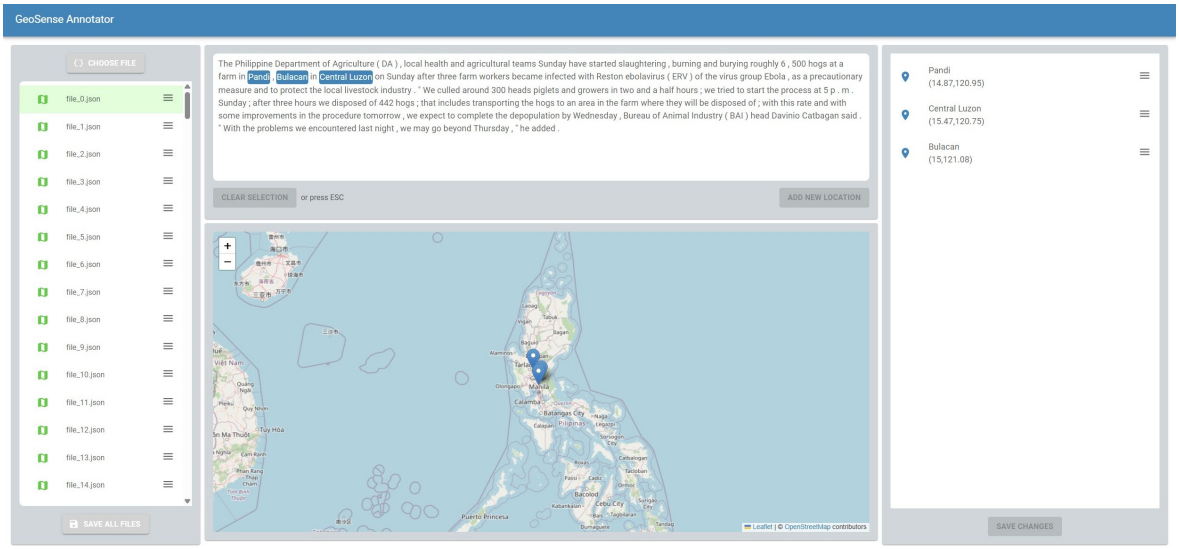


Abbildung 2.1. Benutzeroberfläche des GeoSense Annotators

Wird auf einen der Listeneinträge geklickt, erscheinen die der Datei zugehörigen Daten auf den weiteren Flächen. In diesem Fall erscheinen der Text auf der zentral-oberen Fläche, die vorgelegten Standorte auf der rechten Seite, die dem Text zu entnehmen sind, und die exakten Positionen auf einer virtuellen Karte auf der zentral-unteren Fläche. Die Positionen werden als blaue Markierungen abgebildet, zu denen per Klick die Ortsnamen und die genauen Koordinaten (Breiten- und Längengrad) angezeigt werden. Die virtuelle Karte ist eine React-Komponente aus einer Bibliothek, so genannt React-Leaflet [7], die eine vorgefertigte Version einer Karte bereitstellt, zu der man zahlreiche mögliche Anpassungen vornehmen kann. Auf der Liste der Georeferenzen können nach dem CRUD [8]-Prinzip beliebig Änderungen vorgenommen werden, d.h. das Hinzufügen, das Bearbeiten oder das Löschen eines Eintrages.

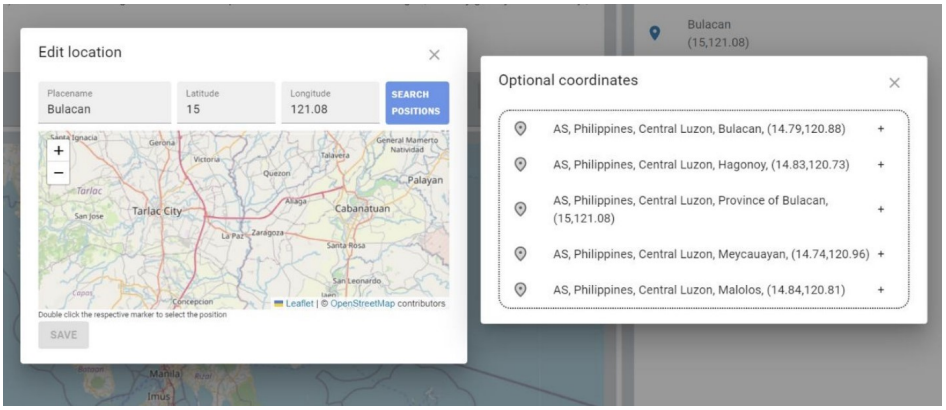


Abbildung 2.2. Dialog des GeoSense Annotators

Bei der Add- und Edit-Funktionalität erscheint ein Dialog (siehe Abbildung 2.2), auf dem die Koordinaten manuell eingetragen werden können oder je nach der eingetragenen Ortsbezeichnung („Placename“)

[7] https://react-leaflet.js.org/
[8] Create, Read, Update, Delete

optionale Koordinaten aus der Backend-Schnittstelle geladen werden können. Beim Betätigen des „Search positions“-Knopfes wird ein API-Aufruf an die API-Schnittstelle im Backend geschickt, die über die gesendete Abfrage mit dem Ortsnamen fünf optionale Koordinaten mittels des „Geocoder“-Frameworks aus der OpenStreetMap-Datenbank ausfindig macht.

2.4 Bisherige Ansätze und aktueller Stand der Forschung

Der Prozess der Georeferenzierung umfasst verschiedene Ansätze. Diese Ansätze lassen sich unterteilen in vier Kategorien [8, vgl. Kapitel 3.1][1, vgl. Kapitel 2]: Regelbasierte, Gazetteer-basierte, lernbasierte und hybride Ansätze.

Regelbasierte Ansätze wenden vordefinierte Regeln und Muster an, um Ortsinformationen aus Texten zu extrahieren. Solche Regeln können auf grammatikalischen Merkmalen eines Satzes basieren, wie dem Vorkommen bestimmter Schlüsselwörter oder Satzstrukturen, oder Mustervergleiche und lexikalische Analysen umfassen. Ein einfaches Beispiel wäre die Anwendung von regulären Ausdrücken, um Ortsnamen zu identifizieren. Dieser Ansatz ist besonders nützlich, wenn die zu extrahierenden Informationen klar definiert und konsistent sind. Allerdings sind rein regelbasierte Ansätze seit dem Aufkommen von lernbasierten Ansätzen selten geworden. Ein Vorteil des regelbasierten Ansatzes ist seine Präzision bei der Erkennung spezifischer Muster. Allerdings kann er bei komplexeren oder mehrdeutigen Texten an seine Grenzen stoßen, da er nicht die Flexibilität und Lernfähigkeit von maschinellen Lernmodellen besitzt.

Der lernbasierte Ansatz beruht auf die Nutzung maschineller Lernalgorithmen, um Ortsinformationen aus Texten zu identifizieren und zu extrahieren. Diese Algorithmen werden auf einem großen Korpus von Textdaten trainiert und lernen, auf Grundlage von Mustern in den Daten Ortsinformationen zu erkennen. Lernbasierte Ansätze sind besonders effektiv bei der Verarbeitung komplexer oder mehrdeutiger Textdaten, die für regelbasierte Ansätze schwer zu erkennen sind. Ein Vorteil des lernbasierten Ansatzes ist seine Flexibilität und Anpassungsfähigkeit an verschiedene Textarten und Domänen. Allerdings erfordert er eine große Menge an annotierten Trainingsdaten und kann rechnerisch intensiv sein.

Bei dem Gazetteer-basierten Ansatz, auch genannt Gazetteer-Matching (Englisch für Ortsverzeichnisabgleich), werden die Ortsinformationen in den Textdaten mit einer bereits vorhandenen Standortdatenbank, sogenannten Gazetteers, wie GeoNames [9] oder OpenStreetMap [10] verglichen. Diese Datenbanken enthalten umfangreiche Informationen zu Ortsnamen und deren geographischen Koordinaten. Bei diesem Ansatz werden zunächst potentielle Ortsnamen aus dem Text extrahiert. Diese Ortsnamen werden dann mit den Einträgen im jeweiligen Gazetteer durch einfache Wort-Vergleiche oder durch fortgeschrittene Techniken abgeglichen, um Tippfehler oder andere Schreibweisen zu berücksichtigen [3, vgl. Kapitel 4.3]. Wenn ein Ortsname im Gazetteer gefunden wird, werden die entsprechenden geographischen Koordinaten diesem Ortsnamen zugewiesen. Ein Vorteil des Gazetteer-Matchings ist seine Genauigkeit bei der Zuordnung von Ortsnamen zu spezifischen geographischen Koordinaten. Allerdings kann es bei mehrdeutigen, unvollständigen Ortsnamen oder bei fehlenden Einträgen im Gazetteer zu Herausforderungen kommen.

Die Elemente aus den verschiedenen Ansätzen werden im hybriden Ansatz zusammengefasst. Dieser

[9] https://www.geonames.org/
[10] https://www.openstreetmap.org/

Ansatz nutzt die Präzision und Einfachheit von regelbasierten Methoden zusammen mit der Flexibilität und Anpassungsfähigkeit von maschinellen Lernmodellen. Regeln werden verwendet, um einfache und eindeutige Ortsnamen zu identifizieren, während maschinelle Lernmodelle komplexere oder mehrdeutige Fälle behandeln. Kontextuelle Informationen werden auch genutzt, um die Genauigkeit der Georeferenzierung zu verbessern. Umliegende Wörter und Sätze werden mitberücksichtigt, um die Bedeutung und Relevanz eines potentiellen Ortsnamens besser zu verstehen. Durch die Kombination dieser verschiedenen Techniken kann der hybride Ansatz eine höhere Genauigkeit und Robustheit bei der Georeferenzierung erreichen, insbesondere in komplexen oder mehrdeutigen Texten.

2.4.1 Geoparser

Geoparser sind Softwaretools, die geographische Entitäten in Texten erkennen und diese mit geographischen Koordinaten verknüpfen. Sie kombinieren Techniken der Named Entity Recognition (NER) mit *Geocoding* [20], um genaue Orte zu identifizieren und zu verorten [4, vgl. Kapitel 3.2]. Der Prozess des Geoparsing-Algorithmus sieht wie folgt aus [21, vgl. Kapitel 2][1, vgl. Kapitel 1]

1. Der unstrukturierte Text wird in den Geoparser eingepflegt, z.B. in Form einer einfachen Benutzeroberfläche.
2. ***Geotagging***: Mit der integrierten NER-Technik werden geographische Ortsnamen in dem Text identifiziert und disambiguiert, also kontextualisiert.
3. ***Geocoding***: Die Koordinaten, also der Breiten- und Längengrad, werden den identifizierten Ortsnamen zugewiesen. Es erfolgt also die Verknüpfung der Entitäten mit Gazetteers.
4. Die ermittelten Ortsnamen und die zugehörigen Koordinaten werden ausgegeben.

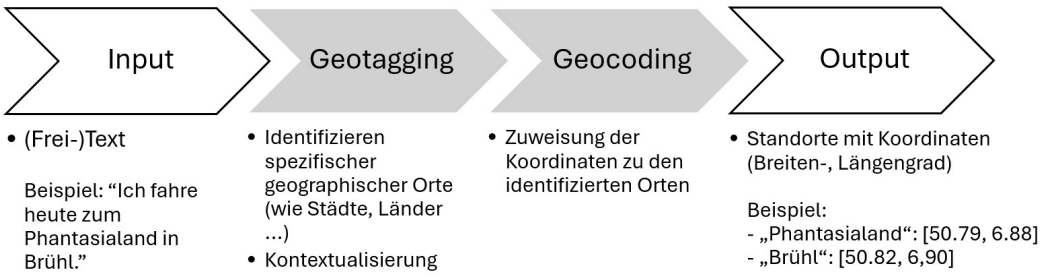


Abbildung 2.3. Ein beispielhafter Geoparsing-Prozess

Traditionell nutzen Geoparser eine Kombination aus Regelwerken und Gazetteers, um geographische Namen in Texten zu erkennen und deren Positionen zu bestimmen. Diese Ansätze sind zwar zuverlässig für strukturierte Texte, stoßen aber bei unstrukturierten oder variantenreichen Daten an ihre Grenzen [1, vgl. Kapitel 2]. Fortschrittliche Geoparser setzen maschinelle Lernverfahren ein, insbesondere neuronale Netzwerke und Transformer-Modelle, um Kontexte besser zu verstehen und präzisere Georeferenzierungen durchzuführen [22, vgl. Kapitel 2]. So nutzt *CamCoder*, entwickelt von Gritta et al. [21], ein Modell, das auf konvolutionalen neuronalen Netzwerke [23] basiert. Dieser Geoparser nutzt lexikalisches als auch geographisches Wissen, um geographische Informationen in einem Text zu identifizieren und zu disambiguieren.

Der Cartographic Location And Vicinity INdexer [22, vgl. Kapitel 2] [4, vgl. Kapitel 3.2] [24, vgl. Kapitel 3.3] (kurz: *CLAVIN* [11] und auf Deutsch „Kartographischer Standort und Umkreis Indexer“) hingegen kombiniert traditionelle Gazetteer-basierte Ansätze mit maschinellen Lernverfahren, indem er frei zugängliche Tools zur Verarbeitung natürlicher Sprache nutzt, unscharfe Suchalgorithmen anwendet, um mit falsch geschriebenen Ortsnamen besser umzugehen, und auf eine geographische Datenbank zugreift. Durch diese Kombination ist CLAVIN in der Lage, Ortsnamen zuverlässig zu identifizieren und ihre Positionen zu bestimmen, selbst wenn diese unterschiedlich geschrieben sind oder in komplexen sprachlichen Kontexten auftreten.

Mordecai [12] [25, 26] ist ein Geoparser, der ebenso auf traditionelle Gazetteer-basierte Ansätze mit maschinellen Lernverfahren setzt und ein integriertes neuronales Netzwerk nutzt. Er wurde entwickelt, um mehrere Funktionen zu bieten, die in herkömmlichen Geoparsern fehlen, darunter eine bessere Handhabung von Ortsnamen, die nicht aus den USA stammen, eine portable Einrichtung einer REST-API-Architektur und eine einfache Anpassung mit Python. Die technische Schlüsselkomponente der früheren Version von Mordecai ist die sprachunabhängige word2vec Architektur von Mikolov et al. [27], die die Ableitung des korrekten Landes für eine Reihe von Orten in einem Textstück ermöglicht. Zur Erkennung von geographischen Entitäten wird in der aktuellsten Version ein von spaCy [13] bereitgestelltes NER-Modell verwendet. spaCy ist eine frei zugängliche Bibliothek für fortgeschrittene Verarbeitung natürlicher Sprache in Python. Identifizierte geographische Entitäten werden in dem GeoNames-Gazetteer mithilfe der Suchmaschine Elasticsearch nach übereinstimmenden Einträgen abgefragt und gegebenenfalls mit präzisen Koordinaten verknüpft. Durch die Kombination dieser Technologien eignet sich Mordecai für Szenarien, in denen eine präzise Erkennung und Zuordnung geographischer Entitäten bei unstrukturierten und variantenreichen Texten gefragt ist.

Die begrenzte Auswahl der Geoparser erfolgte bewusst, um zentrale Konzepte und Technologien, die den aktuellen Stand der Geoparsing-Methoden beschreiben, prägnant darzustellen. Ein umfassenderer Einblick in weitere Geoparser wäre außerhalb des Umfangs und könnte die Struktur der Arbeit verwässern. Die vorgestellten Beispiele sollen eine wesentliche Abdeckung der Funktionalität und Stärken in diesem Bereich repräsentieren.

2.4.2 Geo-Annotationstools

Aufgrund der Knappheit von geographischen Datensätzen haben Forscher aus Internetquellen Daten automatisch extrahiert und in einem Korpus gesammelt. Wie Karimzadeh et al. [28, vgl. Kapitel 2.2] erläutern, hat Gritta et al. [4] eine Liste mehrdeutiger Ortsnamen aus dem GeoNames-Gazetteer erstellt, Duplikate dieser Ortsnamen entfernt und Wikipedia-Einträge entsprechend der verbleibenden Namen in der Liste abgerufen. So stellt der erste Absatz eines Wikipedia-Eintrages einen Datensatz im Korpus dar, wobei der Ortsname aus der ursprünglichen Liste der einzige, annotierte Ortsname ist. Nach diesem ähnlichen Verfahren hat Ju et al. [29] einen Korpus erstellt, welches aus 5500 Sätzen besteht. Mehrdeutige Ortsnamen wurden aus einer Wikipedia-Liste[14] mit den häufigsten US-amerikanischen Ortsnamen bezogen und Internetseiten, die diese Ortsnamen beinhalteten, wurden in einem Korpus gesammelt. Die erstellten Korpora sind zwar gut für die Entwicklung von Geoparsern, wie Karimzadeh et al. argumentieren, allerdings können die Korpora, die ebenso

[11] https://github.com/bigconnect/clavin
[12] https://github.com/openeventdata/mordecai
[13] https://spacy.io/
[14] https://en.wikipedia.org/wiki/List_of_the_most_common_U.S._place_names

auf eine Auswahl von Fehlern ausgerichtet sind, negative Auswirkungen beim Training eines Geoparsers, der auf den lernbasierten Ansatz beruht, haben. Außerdem können die aus dem Internet gewonnenen geographischen Informationen in den Gazetteers nicht gleichermaßen vertreten sein [30]. Um ein qualitatives Korpus mit unverzerrten geographischen Informationen zu ermöglichen, die beispielsweise in Bezug auf die räumliche Abdeckung, die Arten von Orten oder die sprachliche Vielfalt akkurat sind, ist ein gut konzipiertes Annotationstool für eine genaue Annotation von Vorteil.

So wurde von Leidner im Jahre 2007 [31] das vermutlich erste Geo-Annotationstool entwickelt, das es Nutzern ermöglichte, zuvor identifizierte Ortsnamen im Text manuell in speziellen Gazetteers in Toponyme aufzulösen. Toponyme sind Ortsnamen, die von ihren topographischen, also physischen Merkmalen der Erdoberfläche abgeleitet wurden, wie Straßen, Gebäude oder Berge und Täler. Der webbasierte *Toponym Annotation Markup Editor* (auf Deutsch: „Editor für Toponym-Annotationen“ und kurz: TAME) verwendet eine Kombination aus linguistischen Heuristiken und einem fest integrierten Gazetteer, um Toponyme einschließlich lokaler Muster und räumlichem Wissen zu lösen. Die frühere Version von TAME wurde in der aktuellen Version TAME II [32] optimiert, indem eine geographische Repräsentation der Toponyme angeboten und die flexible Integration verschiedener Gazetteers ermöglicht wird. Neben Punktkoordinaten werden auch Polygone und minimal umschreibende Rechtecke zur präziseren Darstellung von Orten unterstützt.

Der Edingburgh Geo-annotator ist ein weiteres webbasiertes Geo-Annotationstool, das die Zuordnung von Ortsnamen zu Toponymen in dem GeoNames-Gazetteer ermöglicht [33]. Dieser bietet eine verknüpfte Texte- und Kartenoberfläche, auf der die Nutzer nacheinander die Ortsnamen im Text anklicken und jeweils die Toponyme zuweisen können, die wiederum auf der entsprechenden Kartenansicht angezeigt werden. Zudem sind die Toponyme auf der Kartensicht nicht beschriftet, was die Nutzer dazu auffordert, jeweils auf die Toponyme zu klicken, um den Namen und Typen des Toponyms anzeigen zu lassen. Dies stellt einen potentiell zeitaufwändigen Vorgang dar.

DeLozier et al. [34, vgl. Kapitel 4] zeigt in seiner Arbeit die Entwicklung einer webbasierten Schnittstelle für Geo-Annotation, die unter dem Namen GeoAnnotate [15] veröffentlicht wurde. Dieses Tool wurde speziell für die Annotation eines Teilkorpus des „War of the Rebellion"“ [16] (WOTR) entwickelt. Um Verwechslungen zu vermeiden, bezeichnen Karimzadeh et al. [28, vgl. Kapitel 2.2] es als WOTR GeoAnnotate. Ein wesentlicher Vorteil des Tools liegt in seiner flexiblen Benutzeroberfläche, da sie verschiedene Entitäten wie Ort, Person und Organisation im Text unterscheidet und Nutzern die Möglichkeit bietet, für jeden dieser Typen eine geographische Referenz in Form von Punkt- oder Polygongeometrien hinzuzufügen. Allerdings fehlt die Funktionalität, Ortsnamen direkt in Toponyme aus einem Gazetteer aufzulösen, was zur Folge hat, dass Nutzer die Orte selbständig über das Internet nachschlagen und die entsprechenden Koordinaten finden müssen.

Wie Karimzadeh et al. betonen, bieten die Geo-Annotationstools TAME, Edinburgh Geo-annotator und WOTR Geoannotator keine Funktion zur Korrektur versehentlich annotierter Ortsnamen. Dies führt dazu, dass die Nutzer keine Möglichkeit haben, bereits vorgenommene Annotationen – sei es durch menschliche Annotatoren oder ein automatisches NER-System – vor der Toponym-Auflösung (Zuordnung der Namen zu spezifischen Orten) durch dynamische Abfragen im Gazetteer zu bereinigen. Eine solche Korrekturmöglichkeit wäre jedoch wichtig, um die Qualität und Präzision des annotierten Datensatzes zu gewährleisten.

[15] https://github.com/utcompling/GeoAnnotate/
[16] http://ehistory.osu.edu/books/official-records

So haben Karimzadeh et al. den GeoAnnotator [17] entwickelt, der nicht nur eine kollaborative Plattform zur semi-automatischen Annotation mit integrierter Text- und Kartenansicht bietet, sondern auch automatische Vor-Annotationen generieren und mehreren Nutzern die gleichzeitige Erstellung und Korrektur annotierter Datensätze ermöglichen kann. Der GeoAnnotator baut auf das skalierbare Geoparsing-System GeoTxt [35] auf, welches sechs verschiedene NER-Algorithmen zur Erkennung von Ortsnamen bietet und ein leistungsfähiges Suchmaschinen-Framework zur Toponym-Auflösung verwendet.

Höhn et al. [36] entwickelten den Referencing and Annotation Tool (kurz: RAT und auf Deutsch „Referenzierungs- und Annotationstool“), der darauf abzielt, Ortsmarkierungen in digitalisierten historischen Karten zu identifizieren und zu georeferenzieren. Das System verwendet Bildverarbeitungsalgorithmen, um Symbole und Beschriftungen, die Orte auf historischen Karten darstellen, automatisch zu identifizieren. Die identifizierten Orte werden anschließend mit modernen geographischen Koordinaten verknüpft und können bei Bedarf korrigiert werden. Obwohl RAT eine gute Grundlage für die semi-automatische Annotation geographischer Informationen bietet, ist es dennoch nur für die Verarbeitung von historischen Karten konzipiert und bietet keine Möglichkeit zur Annotation von Ortsnamen in Texten.

Darüber hinaus gibt es eine kostenpflichtige Plattform für umfassendere Datenannotationen, so genannt Kili Technology [18], die verschiedene Datentypen unterstützt, einschließlich geographischer Daten. Kili Technology unterstützt die Annotation von Ortsnamen in Texten und ähnlich wie bei dem Annotationstool RAT, wird die Annotation von georeferenzierten Rastergraphiken ermöglicht. Dabei werden Kartenansichten von herkömmlichen Gazetteers wie OpenStreetMap als Hintergrundlayer verwendet, um die Positionierung von Annotationen zu erleichtern. Des Weiteren können die Distanzen der verschiedenen Punktpositionen für genauere Analysen gemessen werden. Statt integrierte Gazetteers oder NER-Systeme zu verwenden, ermöglicht die Plattform die Integration externer LLMs, um spezifische Aufgaben wie die Georeferenzierung zu erfüllen. Obwohl die Software die Nutzung und das iterative Training eines LLMs ermöglicht, erfordert dies jedoch zusätzliche Implementierungsschritte seitens der Nutzer.

Trotz ihrer Funktionalität setzen die genannten Tools, mit Ausnahme von Kili Technology, nicht auf den Einsatz moderner LLMs. Die potentiellen Vorteile, die LLMs bieten können, beispielsweise Mehrdeutigkeiten in Ortsnamen besser kontextbasiert zu interpretieren und aufzulösen, bleiben so ungenutzt. Die ausschließliche Verwendung von Gazetteers bringt zudem Einschränkungen mit sich. Unvollständige oder veraltete Einträge in den Gazetteers können die Präzision der Georeferenzierung beeinträchtigen. Auch die Genauigkeit der Punktpositionierung komplexer geographischer Merkmale, wie Flüsse oder Gebirgsketten, wird von der fehlenden Berücksichtigung kontextbezogener Bedeutungen beeinträchtigt [28, vgl. Kapitel 2.2] [31, vgl. Kapitel 2.3]. Die Integration von LLMs könnte umfassendere Textanalysen ermöglichen und auf umfangreichere Sprachressourcen und Kontextinformationen zurückgreifen.

[17] https://github.com/geovista/GeoTxt
[18] https://kili-technology.com/

Kapitel 3

Konzept

3.1 Auswahl und Training der LLMs

Die Idee bei der Auswahl des jeweiligen LLMs bestand darin, dies dem Nutzer zu überlassen (siehe Abbildung A.2). Das heißt, dass auf der Benutzerschnittstelle die Möglichkeit angeboten werden soll, zwischen den verschiedenen Providern (Englisch für Anbieter) zu wechseln und die Geoparsing-Aufgabe mit dem ausgewählten Provider auszuführen (siehe Abbildungen A.9 und 4.13). Der Nutzer soll in der Lage sein, die erforderlichen Daten für den Provider anzugeben, wie die Adresse des Servers, worauf das LLM gehostet wird, das anzuwendende Sprachmodell oder der API-Schlüssel, beispielsweise um den Zugriff auf das ausgewählte Modell von OpenAI zu ermöglichen. Im Backend werden sowohl die Daten der Provider lokal gespeichert, um unerlaubten externen Zugriff zu verhindern, als auch API-Schnittstellen zur Abfrage und Nutzung der jeweiligen Provider bereitgestellt, die wiederholt abrufbar sind (siehe Abbildung A.3).

Allerdings eignen sich nicht alle Sprachmodelle für den Geoparsing-Prozess. Die Sprachmodelle von OpenAI und LLaMA sind besonders gut darin, Texte zu generieren und zu vervollständigen. Diese Fähigkeiten ermöglichen es den Modellen, geographische Entitäten im Kontext zu identifizieren und zuzuordnen. Durch die Anwendung des Mechanismus „Byte Pair Encoding“ (BPE) [12, vgl. Kapitel 3.9.2] [13, vgl. Kapitel 2.1] sind diese Modelle außerdem in der Lage, mit Wörtern außerhalb ihres Vokabulars umzugehen. Unbekannte Wörter werden in kleinere Einheiten zerlegt, wodurch seltene oder komplexe geographische Begriffe effektiver aufgegriffen werden können [37]. Während der bidirektionale Ansatz von BERT hervorragend darin ist, spezifische Entitäten in Texten zu erkennen und den unmittelbaren Kontext zu verstehen, ist es nicht primär auf Textgenerierung oder -vervollständigung ausgerichtet und kann bei der Verarbeitung komplexer, unstrukturierter Daten und der Zuordnung von geographischen Informationen an seine Grenzen stoßen [11, vgl. Kapitel 1 und 3]. Daraus lässt sich ableiten, dass BERT weniger für die Georeferenzierung geeignet ist.

Das Training von LLMs für die Georeferenzierung erfordert in der Regel eine sorgfältige Auswahl der Trainingsdaten und eine Feinabstimmung der Modellparameter. Ein gängiger Ansatz ist die Verwendung von vortrainierten Modellen, in unserem Fall von OpenAI oder LLaMA, die dann auf die Geoparsing-Schritte abgestimmt werden. Der Trainingsprozess ist wie folgt gegliedert:

1. **Datensammlung**: Sammlung großer Textmengen, die geographische Informationen erhalten.

2. **Vorverarbeitung**: Bereinigung und Annotation der Daten, um sicherzustellen, dass sie für das Training

geeignet sind.

3. **Feinabstimmung**: Anpassung des Modells an die spezifische Aufgabe der Georeferenzierung durch weiteres Training auf den vorbereiteten Daten.

In unserem Fall orientieren wir uns am Prinzip des Active Learnings [38, S. 3f, Kapitel 1.1]. Das Active Learning ist ein Lernverfahren im maschinellen Lernen, insbesondere in Bezug auf die iterative Verbesserung von Modellen durch gezielte Datenauswahl und menschlichem Feedback. Das heißt, dass Datenmengen sowohl gezielt gesammelt als auch vom Nutzer, welche er im Frontend manuell annotiert und korrigiert, bereitgestellt werden. Die Datensätze werden in einem implementierten Feedback-Mechanismus gespeichert und fortgehend überwacht. Sobald ein Schwellenwert an Datensätzen erreicht wird, wird das Retrain-Job, also der Nachtrainierungsprozess des Modells gestartet und das Modell auf den Daten fein-abgestimmt. Dieser iterative Prozess ermöglicht es, das Modell kontinuierlich zu verbessern und potentielle Ausgabefehler beim Geoparsing-Prozess zu minimieren.

Für die semi-automatische Annotation nutzen wir das Modell *LLaMA 3.1 8B Instruct* [1]. Dieses Modell bietet mehrere wesentliche Vorteile gegenüber anderen Modellen wie die GPT-Modelle von OpenAI. Zum einen ist es Open-Source, was bedeutet, dass es nicht den gleichen Beschränkungen und Nutzungskosten unterliegt wie die modernen GPT-Modelle und frei zugänglich ist. Zum anderen liegt der weitere wesentliche Vorteil in seiner Ressourceneffizienz [13, vgl. Kapitel 1 und 3]. Während ChatGPT für viele Aufgaben geeignet ist, erfordert es deutlich mehr Rechenleistung und Speicherressourcen. LLaMA hingegen ist speziell für vergleichbare Rechenressourcen oder sogar je nach Feinabstimmung des Modells für bessere Ergebnisse optimiert. Dies ist besonders wichtig, da auf begrenzter Infrastruktur entwickelt und experimentiert wird. Der Unterschied zwischen dem Instruct-Modell und dem Basis-Modell *LLaMA 3.1 8B* ist die verbesserte Leistung des Instruct-Modells durch zusätzliche Feinabstimmung für konversationsorientierte Aufgaben, die dialog- und assistentbasierte Szenarien erfordern. Abgesehen von der Ressourceneffizienz und der Kostenfreiheit eignet sich das Modell besonders gut für das Active Learning, da es kontinuierlich verbessert und verfeinert werden kann, indem es mit menschlichem Feedback trainiert wird. Dies ermöglicht die schrittweise Verbesserung der Genauigkeit des Geoparsing-Prozesses, was bei der Verarbeitung von großen Datensätzen von Vorteil ist.

3.2 Entwicklung der Benutzeroberfläche

Wie in Kapitel 2.3 *GeoSense Annotator* erwähnt, habe ich das Tool GeoSense Annotator zur Annotation von (Web-)Daten mit Georeferenzen im Rahmen meines Praxisprojektes entwickelt. Allerdings erfolgt die Annotation der Daten nicht semi-automatisch, sondern vollständig manuell. Das heißt, dass jeweils auf die Ortsnamen geklickt und die Koordinaten aus dem OpenStreetMap-Gazetteer im Backend bezogen oder manuell eingegeben werden müssen (siehe Abbildung A.8). Um jedoch die Ortsnamen automatisch zu extrahieren und die Koordinaten zuzuweisen, also das Geoparsen von (Web-)Daten zu ermöglichen, wurde ein „Geoparse“-Button im Textinhalt-Raster eingepflegt, worüber der Textinhalt über eine API-Schnittstelle an das Backend übermittelt wird. Die Daten werden schließlich an den im Frontend konfigurierten Provider gesendet und weiterverarbeitet. Beliebig viele Provider können auf der separaten Provider-Seite des GeoSense Annotators hinzugefügt und ausgewählt werden, zu denen jeweils Daten wie der Verbindungsstatus, der selektierte Konversationsstil und die Anzahl bearbeiteter Datensätze zusätzlich auf einem separaten Raster angegeben werden.

[1] https://huggingface.co/meta-llama/Meta-Llama-3.1-8B-Instruct

Sobald der Geoparsing-Prozess erfolgt ist, werden die generierten Georeferenzen an das Frontend gesendet und auf einem externen Dialog angezeigt. Auf dem Dialog kann der Nutzer schließlich entscheiden, ob er die alten Georeferenzen mit den neuen Georeferenzen überschreiben möchte. Falls der Nutzer sich dazu entschieden hat, die neuen Georeferenzen zu übernehmen, werden diese auf der Georeferenzen-Liste angezeigt und die jeweiligen Ortsnamen im Text des Textinhalt-Rasters markiert. Andernfalls kann er die neuen Daten verwerfen und mit dem Prozess der manuellen Annotation fortfahren.

3.3 Integration von MLflow

MLflow[2] ist eine Open-Source-Plattform, die für das Training und die Verfolgung von Machine-Learning-Prozessen entwickelt wurde. Es erleichtert die Verwaltung von Machine-Learning-Projekten und die damit einhergehenden Herausforderungen, wie die Reproduzierbarkeit von Experimenten, die Verfolgung von Modellentwicklungen und die einfache Bereitstellung trainierter Modelle in Produktions- und Testumgebungen. Die Verwendung von MLflow kann dabei helfen, das Training der LLM-Modelle systematisch zu verfolgen. Dies kann geschehen, indem alle wichtigen Parameter, Metriken, Modelle und andere Artefakte während des Trainings von MLflow aufgezeichnet werden. Die Integration erfolgt typischerweise in den folgenden Schritten:

1. **Tracking von Modellen**: Die Trainingsverläufe der Modelle, also die Parameter und Metriken jedes Trainingslaufs, werden gespeichert, wie zum Beispiel Lernrate, Genauigkeit und Verluste, was für die Analyse und Verbesserung der Modelle von Bedeutung ist. Diese Daten dienen im weiteren Verlauf dazu, die Modelle nach den gemessen Parametren zu ordnen sowie den Retrain-Job-Mechanismus im Backend des Projektes gegebenenfalls anzupassen. Außerdem werden die Evaluationen der produzierten Annotationen und das menschlichen Feedback gespeichert, sodass die Präzision und Genauigkeit der Modelle ausgewertet werden können.

2. **Modellregistrierung und -versionierung**: Nach dem Training werden die Modelle in einem zentralen Modellregister gespeichert und versioniert. Dies ist besonders nützlich, um den besten Modellstand zu identifizieren und diese für den GeoSense Annotator verfügbar zu halten.

3. **Experimentanalyse**: MLflow bietet eine Web-Benutzeroberfläche an, die dem Entwickler ermöglicht, alle aufgezeichneten Experimente einzusehen und zu analysieren. So können die besten Modelle identifiziert werden, um Fehler, die zu einer schlechten oder fehlerhaften Modellleistung führen könnten, zu vermeiden.

4. **Modellbereitstellung**: Trainierte Modelle können mit MLflow in Umgebungen integriert werden, entweder durch die Bereitstellung als REST-API oder durch die Integration in bestehenden Systemen. In unserem Fall würde das Hostserver-System, welches wir für den Geoparsing-Prozess im Backend nutzen, auf das Modellregister und somit auf den Pfad des gespeicherten Modells zugreifen.

[2] https://mlflow.org/docs/latest/introduction/index.html

Kapitel 4

Implementierung

In diesem Kapitel wird die Implementierung des entwickelten Systems detailliert beschrieben. Dabei wird zunächst auf die Benutzeroberfläche und die verwendeten Technologien eingegangen, gefolgt von einer Beschreibung der einzelnen Komponenten. Besonderes Augenmerk liegt auf der Integration von den LLMs zur semi-automatischen Annotation von Webdaten mit Georeferenzen sowie der Nachverfolgung des Trainings eines LLMs mit dem Framework MLflow, was unter anderem die Protokollierung und den direkten Vergleich von Evaluations- und Trainingsmetriken verschiedener Trainingsiterationen erleichtert. Im Folgenden werden die Implementierungsschritte im Detail erläutert.

4.1 Frontend-Implementierung

Wie bereits in Kapitel 3.2 erwähnt, wurde die zentral-obere Fläche des GeoSense Annotators, also der Bereich, worauf der Textinhalt abgebildet wird und die manuelle Annotation von Georeferenzen erfolgt, um einen zusätzlichen Knopf, dem „Geoparse“-Button, ergänzt (siehe Abbildung 4.1). Außerdem wurde ein zusätzliches Feld integriert, um die manuelle Selektion eines Ortsnamen wörtlich wiederzugeben.
Um den „Geoparse“-Button betätigen zu können, muss ein Provider auf der Provider-Seite des GeoSense Annotators ausgewählt werden. Auf der Seite können bereits konfigurierte Provider geladen und in der Tabelle angezeigt oder neue Provider hinzugefügt werden (siehe Abbildung 4.2). Bestehende Provider können über die Betätigung des „Load provider“-Buttons geladen werden.

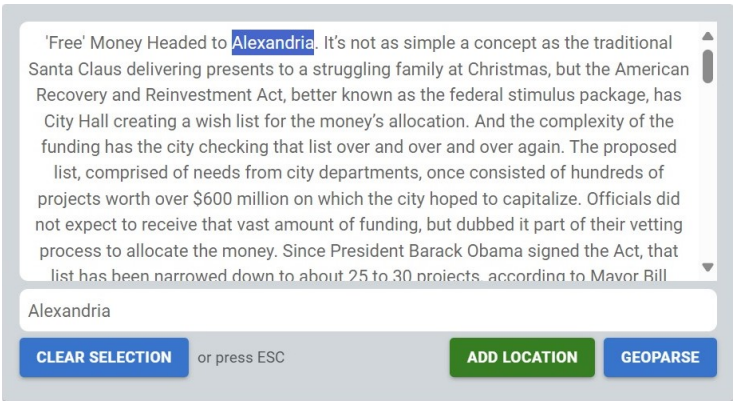


Abbildung 4.1. Textraster des GeoSense Annotators

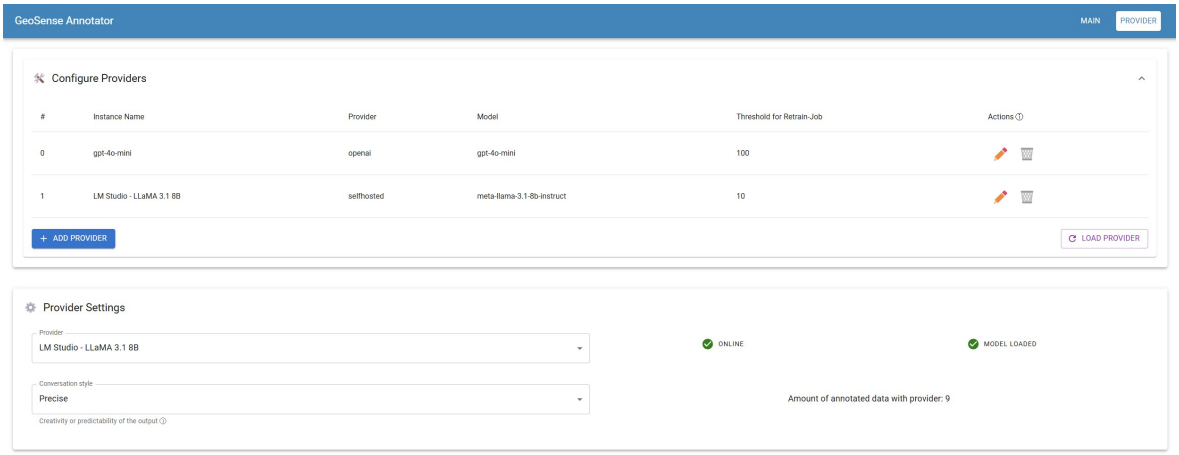
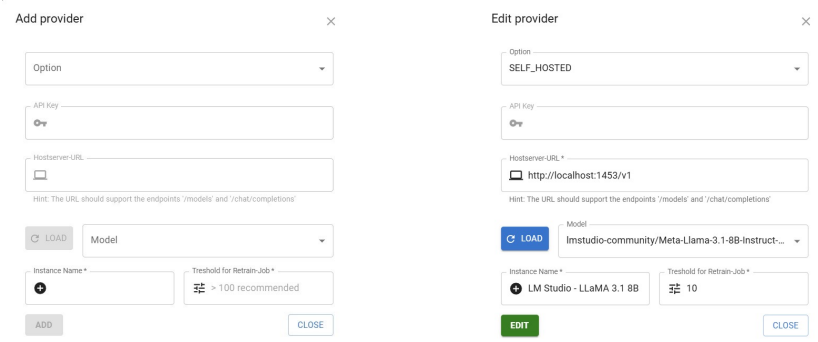


Abbildung 4.2. Provider-Seite des GeoSense Annotators

Wird auf den „Add provider“-Button oder auf den Stift-Symbol in der entsprechenden Tabellenzelle eines bestehenden Providers geklickt, erscheint ein Dialog, worauf ein Provider konfiguriert werden kann (siehe Abbildung 4.3).



(a) Hinzufügen eines Providers **(b)** Editieren eines Providers

Abbildung 4.3. Provider-Dialog des GeoSense Annotators

Es werden zwei Optionen von Providern angeboten. Bei der ersten Option handelt es sich um die Nutzung eines OpenAI-Modells. Hierfür wird ein API-Schlüssel gefordert, um den Zugriff auf die Modelle und die Inferenz mit einem gewählten Modell zu ermöglichen. Diese Variante ist jedoch gebührenpflichtig und kann abhängig vom gewählten Modell und den verarbeiteten Datenmengen kostenintensiv werden. Die zweite Option bezieht sich auf die Nutzung eines selbstgehosteten Providers. Bei dieser Option wird gefordert, dass die Hostserver-Adresse die Endpunkte „/model“ und „/chat/completions“ unterstützt, damit die gehosteten Modelle geladen und die Daten verarbeitet werden können. Wurden eine Option ausgewählt und die erforderlichen Felder ausgefüllt, muss eine Bezeichnung für die Instanz eingegeben und ein Schwellenwert für das Retrain-Job des Providers festgelegt werden. Im unteren Raster der Provider-Seite („Provider Settings“, siehe Abbildung 4.2) kann daraufhin ein bestehender oder neu-hinzugefügter Provider selektiert und die Anzahl der bisher annotierten Datensätze eingesehen werden. Zusätzlich wird für einen selektierten selbstgehosteten Provider der Server- und Modell-Status in Form eines rötlichen oder grünen Symbols mit einer Statuserläuterung angezeigt. Zudem kann der Unterhaltungsstil mit dem Provider festgelegt werden, um die Ausgabe des Providers anzupassen.

Hierfür bieten sich folgende drei Optionen:

- **Precise**: Dies ist der Standardwert. Die Ausgabe ist sehr deterministisch. Es werden Vorhersagen getroffen, die sicherer und präziser sind. Dieser Stil ist nützlich, wenn faktenbasierte Ausgaben erwartet werden.

- **Balanced**: Die Ausgabe ist ausgewogen divers. Das bedeutet, dass eine Balance zwischen Determinismus und Kreativität gefunden wird. Dies hat zur Folge, dass vielfältige, aber sinnvolle Ausgaben erwartet werden.

- **Creative**: Die Ausgaben sind kreativer, variantenreicher und weniger vorhersehbar. Allerdings kann es dazu führen, dass die Ausgaben weniger sinnvoll sind. Daher wird diese Option nur zu Experimentierzwecken empfohlen.

Sobald der „Geoparse“-Button betätigt wird, erscheint ein Dialog mit einer Liste von Georeferenzen und eine Darstellung der Georeferenzen auf einer virtuellen Karte (siehe Abbildung 4.4). Der Nutzer kann in diesem Fall entscheiden, ob die bestehenden Annotationen durch die Betätigung des „Save“-Buttons mit den neuen Annotationen überschrieben oder die ermittelten Georeferenzen durch das bloße Verlassen des Dialogs verworfen werden. Werden die Georeferenzen übernommen, kann der Nutzer Korrekturen oder Ergänzungen an den Annotationen vornehmen und diese schließlich speichern. Beim Speichern werden die Daten an das Backend übermittelt und für die Feinabstimmung des jeweiligen Providers lokal weiterverarbeitet.

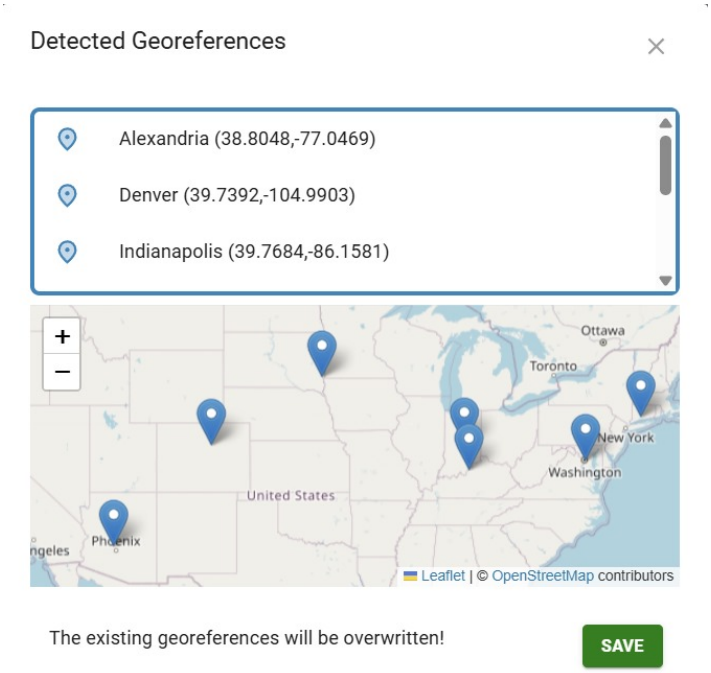


Abbildung 4.4. Geoparse-Dialog des GeoSense Annotators

4.2 Backend-Implementierung

Im Backend des GeoSense Annotators werden die Daten vom Frontend mithilfe von HTTP-Methoden wie POST zur Datenverarbeitung, GET zur Datenabfrage oder DELETE/PUT zur Datenaktualisierung über API-Schnittstellen verarbeitet. HTTP [1] (Hypertext Transfer Protocol) ist das Kommunikationsprotokoll, das den Datenaustausch zwischen Client und Server ermöglicht. Diese Anfragen werden über das FastAPI-Framework konfiguriert. Die Daten werden zur Sicherstellung der Datenintegrität in einem lokalen Ordner des Nutzers gespeichert. Die im Backend verarbeiteten Daten umfassen sowohl die Vorschläge für Georeferenzen, die Informationen der Provider als auch das menschliche Feedback, das für die kontinuierliche Verbesserung des im Provider konfigurierten LLMs genutzt wird. Bei jeder Anfrage wird überprüft, ob die Daten den definierten Standards entsprechen.

```python
@router.get("/feedback")
async def countFeedback(instance_name: str):
    try:
        feedback_data = await load_feedback(instance_name, FEEDBACK_DIR_PATH)

        return { "feedback_count": len(feedback_data) }
    except Exception as e:
        raise HTTPException(
            status_code=500,
            detail=f"Error! Failed to count feedback: {str(e)}"
        )
```

(a) GET-Methode zur Feedback-Datenzählung

```python
@router.post("/feedback")
async def feedback(request: FeedbackRequest):
    try:
        # Create feedback directory if directory doesn't exist
        os.makedirs(FEEDBACK_DIR_PATH, exist_ok=True)
        # Save feedback
        await store_feedback(request, FEEDBACK_DIR_PATH)
        # Check, if Retrain-Job needs to be initiated
        await check_feedback_threshold(request.provider, FEEDBACK_DIR_PATH)

        return { "message": "Feedback processed successfully." }
    except Exception as e:
        raise HTTPException(
            status_code=500,
            detail=f"Error! Failed to process feedback: {str(e)}"
        )
```

(b) POST-Methode zur Feedback-Sicherung und Schwellenwertprüfung

Abbildung 4.5. API-Schnittstellen des Feedback-Algorithmus

Ein wesentlicher Bestandteil des Backends ist die Einbindung eines LLMs, in unserem Fall das *LLaMA 3.1 8B Instruct*, das für die semi-automatische Annotation von Webdaten verwendet wird. Zur Generierung von Vorschlägen für Georeferenzen werden die vom Frontend empfangenen Daten an die entsprechende Hostserver-Adresse des Providers geliefert. Dies erfolgt über eine POST-Anfrage (siehe Abbildung B.1). Anschließend werden die Vorschläge entsprechend formatiert und an das Frontend gesendet. Sobald die Validierung und Verfeinerung durch menschliches Feedback erfolgt ist, wird dies im Backend weiterverarbeitet und zugleich geprüft, ob der für den jeweiligen Provider festgelegter Schwellenwert erreicht wurde.

```python
async def check_feedback_threshold(provider: Provider, DIR_PATH) -> None:
    'Retrain-Job-Check for Threshold'

    file_path = os.path.join(DIR_PATH, f"{provider.instance_name}_feedback.json")

    with open(file_path, "r") as f:
        feedback_data = json.load(f)
        if len(feedback_data) >= provider.data["threshold_retrain_job"]:
            # Evaluate feedback
            await evaluateFeedback(feedback_data)
            # Trigger retrain-job for model
            await retrain_model(feedback_data, provider)
            # Clear feedback-data
            open(file_path, "w").close()
```

Abbildung 4.6. Funktion zur Prüfung des Provider-spezifischen Schwellenwertes

[1] https://developer.mozilla.org/en-US/docs/Web/HTTP

Bei Erreichen des Schwellenwertes erfolgt im ersten Schritt die Evaluation der generierten Vorschläge für Georeferenzen. Es wird anhand des menschlichen Feedbacks, also der überarbeiteten Georeferenzen geprüft, wie präzise die Ausgaben des Providers sind. Hierfür wird ein spezieller Algorithmus von DLR verwendet, der folgende Metriken kalkuliert [39]:

```python
async def evaluateFeedback(feedback_data) -> None:
    'Feedback-Evaluation'

    precision, recall, f1_score, matched_coordinates = await compute_precision_recall_f1(feedback_data, "predictions", "corrections")

    # MLFlow Tracking
    with mlflow.start_run(
        run_name="Feedback-Evaluation",
        tags={"feedback": "evaluation"},
        description="Evaluation of feedback → precision, recall, f1, accuracy for locations in ~ 10km and ~ 161km distances"
    ):
        mlflow.log_metric("Precision", precision)
        mlflow.log_metric("Recall", recall)
        mlflow.log_metric("F1 Score", f1_score)
        mlflow.log_metric("A-at-k-161", round(await calculate_A_at_k(matched_coordinates, 161),2))
        mlflow.log_metric("A-at-k-10", round(await calculate_A_at_k(matched_coordinates, 10),2))
```

Abbildung 4.7. Kalkulation der Metriken und Erfassung mit MLflow

- **Precision**: Diese Metrik gibt an, welcher prozentuale Anteil der erfassten Ortsnamen tatsächlich korrekt ist. Ein hoher Wert würde darauf hinweisen, dass nur wenige falsche Zuordnungen gemacht wurden.

- **Recall**: Hier wird gemessen, wie viele der tatsächlich relevanten Ortsnamen gefunden wurden. Ein hoher Wert zeigt, dass das Modell nur wenige Ortsnamen übersehen hat.

- **F1-Score**: In der semi-automatischen Annotation ist es wichtig, dass das Modell sowohl genaue (Precision) als auch vollständige (Recall) Ergebnisse liefert. Der F1-Score hilft, das harmonische Mittel zwischen Precision und Recall zu finden und die Gesamtleistung des Modells in dieser Hinsicht zu bewerten.

- **Accuracy at k** (A@k): Trotz korrekter Erfassung der Ortsnamen kann es sein, dass die Koordinaten nicht exakt stimmen. Hierfür hilft diese Metrik, die Genauigkeit der Koordinaten (Breiten- und Längengrad) in einem bestimmten Tolenzbereich (Radius k) zu messen. Es wird nach dem Prinzip der Point-Radius-Methode vorgegangen, um die Unsicherheiten bei der Bewertung der Koordinaten zu berücksichtigen. Die Methode beschreibt eine Position durch ein Koordinatenpaar und einen Radius, der den Bereich der Unsicherheit umfasst [40, S. 747 ff.]. Die Position in diesem Fall entspricht der tatsächlichen Position der Ortschaft. Für den Bereich der Unsicherheit werden die Radien 10 km und 161 km verwendet. Der kleinere Radius von 10 km ist typisch für Kleinstädte oder dicht besiedelte Stadtteile in Großstädten und ermöglicht eine präzise Bewertung in eng begrenzten geographischen Bereichen. Der größere Radius von 161 km eignet sich für größere geographische Einheiten, wie z.B. Metropolen oder ländliche Gebiete, in denen Städte oder Dörfer auf größeren Gebieten verteilt sind, sodass die Toleranz für Ungenauigkeiten bei weit entfernten Zielen gemessen werden. Dieser Radius entspricht ungefähr 100 Meilen, was eine gängige Referenzgröße in geographischen Studien ist, wie zum Beispiel in der Arbeit von DeLozier et al. [34]. Dies könnte für internationale Standards relevant sein.

  Um die Distanz zwischen den vorhergesagten und den tatsächlichen Koordinaten zu berechnen, wird die Haversine-Formel verwendet. Diese Formel berechnet die kürzeste Entfernung zwischen zwei Punkten auf einer Kugeloberfläche (wie der Erde), basierend auf den Breiten- und Längengraden. Somit ermöglicht die Haversine-Formel eine genaue Bewertung der Abweichungen in der geographischen

Distanz, die für die Berechnung der Metrik entscheidend ist. Die Formel sieht folgendermaßen aus [41, S. 1 ff.]:

$$d = 2r \cdot \arcsin\left(\sqrt{\sin^2\left(\frac{\Delta\varphi}{2}\right) + \cos(\varphi_1) \cdot \cos(\varphi_2) \cdot \sin^2\left(\frac{\Delta\lambda}{2}\right)}\right)$$

wobei:

- d die berechnete Distanz zwischen den zwei Koordinaten ist,
- r der Erdradius (ca. 6.371 km) ist,
- φ_1, φ_2 die Breitengrade der beiden Koordinaten sind,
- $\Delta\varphi$ die Differenz der Breitengrade ist,
- $\Delta\lambda$ die Differenz der Längengrade ist.

```python
def calculate_distance(coord1, coord2) -> float:
    'Distance betweens two points on earth according to the Haversine formula'

    # Function to calculate distance between two coordinates
    # Convert latitude and longitude from degrees to radians
    lat1, lon1 = radians(coord1[0]), radians(coord1[1])
    lat2, lon2 = radians(coord2[0]), radians(coord2[1])

    # Haversine formula
    dlat = lat2 - lat1
    dlon = lon2 - lon1
    a = sin(dlat / 2)**2 + cos(lat1) * cos(lat2) * sin(dlon / 2)**2
    c = 2 * atan2(sqrt(a), sqrt(1 - a))
    distance = 6371 * c  # Radius of Earth in kilometers
    return distance
```

Abbildung 4.8. Haversine-Formel zur Berechnung der Distanz zweier Koordinaten

Für das Training der LLMs wird ein speziell konfiguriertes Remote-Backend (siehe Abbildung A.6) auf der Plattform für wissenschaftliches Rechnen an der Hochschule Bonn-Rhein-Sieg[2] verwendet, die die notwendigen Hardware-Ressourcen sowie die notwendige Software-Infrastruktur bereitstellt, um das Modell regelmäßig basierend auf den gesammelten Feedback-Daten zu trainieren und anzupassen. Die zugrundeliegende Hardware auf der Server-Schnittstelle *wr14* der Hochschul-Plattform zur interaktiven Entwicklung verfügt über 192 GB Arbeitsspeicher und 480 GB Festplattenspeicher, die für die Speicherung und Verwaltung der Speicher-lastigen Modelle relevant sind. Besonders entscheidend für das effiziente Training sind die vier Nvidia V100[3] Grafikprozessoren (auf Englisch „graphics processing unit“ und kurz: GPU), die jeweils um die 16 GB Speicher verfügen und speziell für rechenintensive Aufgaben wie das Training von KI-Modellen entwickelt wurden. Die GPUs, die über eine spezielle Verbindung (NVLink[4]) mit einer Bandbreite von 300 GB/s verbunden sind und insgesamt 64 GB Speicher umfassen, können durch den schnellen Datenaustausch parallel umfangreiche Berechnungen ausführen. Zusammen mit den leistungsstarken Prozessoren wird eine optimale Umgebung für die anspruchsvollen Trainingsprozesse geschaffen.

Das Training erfolgt mithilfe des Frameworks Unsloth [5] [42], das auf die Verarbeitung und Optimierung großer Sprachmodelle spezialisiert ist. Es ermöglicht eine flexible Anpassung der Trainingspipelines, um

[2] https://wr0.wr.inf.h-brs.de/wr/
[3] https://www.nvidia.com/de-de/data-center/v100/
[4] https://www.nvidia.com/de-de/data-center/nvlink/
[5] https://unsloth.ai/introducing

den unterschiedlichen Anforderungen der Modelle und der Hardware gerecht zu werden (siehe Abbildung 4.9 a)). Ein relevantes Merkmal ist das automatische Datenmanagement, das die Verwaltung großer Datensätze effizient gestaltet, um die Hardwareleistung zu maximieren. Eine besondere Kernfunktion von Unsloth ist außerdem die Anpassung eines LLMs durch das Parameter-Effiziente Feintuning (kurz: PEFT, siehe Abbildung 4.9 b)). PEFT ermöglicht es, eine geringe Anzahl von Parametern des Modells anzupassen, anstatt das gesamte Modell neu zu trainieren [43]. In Verbindung mit QLora (Englisch für „Quantized Low-Rank Adaptation“) [44, S. 1 f.] wird dieser Ansatz noch weiter optimiert. QLora ist eine Methode, den Speicherplatzbedarf eines Modells durch die Quantisierung zu verringern, ohne die Genauigkeit übermäßig zu beeinträchtigen und die Modellleistung beizubehalten. Im Kontext von maschinellem Lernen versteht man unter einer Quantisierung, dass die Darstellung der Modellparameter von den ursprünglichen 32-Bit-Gleitkommazahlen (hohe Präzision) auf eine niedrigere Präzision wie 16-Bit oder 8-Bit-Gleitkommazahlen reduziert werden, wodurch Berechnungen schneller und ressourcenschonender sind. Dies reduziert die Trainingszeit und den Rechenaufwand erheblich, was bei großen Modellen wie *LLaMA 3.1 8B Instruct* von Vorteil ist. Diese Besonderheiten machen die Nutzung von Unsloth bei einer begrenzten Speicher- und Rechenkapazität vorteilhaft.

```python
@router.post("/retrain")
async def retrain_model(request: TrainingRequest):
    try:
        from unsloth import FastLanguageModel

        model_name:str = request.provider['data']['model'].split('/')[0]
        model_path = f"models/{model_name}"

        model, tokenizer = FastLanguageModel.from_pretrained(
            model_name = model_path,
            dtype = None,
            load_in_4bit = True,
            max_seq_length = 2048
        )
```

(a) POST-Methode zum Trainieren der jeweiligen LLM

```python
model = FastLanguageModel.get_peft_model(
    model,
    r = 16,
    target_modules = ["q_proj", "k_proj", "v_proj", "o_proj",
                      "gate_proj", "up_proj", "down_proj"],
    lora_alpha = 16,
    lora_dropout = 0.1,
    bias = "none",
    use_gradient_checkpointing = "unsloth",
    random_state = 3407,
    use_rslora = False,
    loftq_config = None,
)
```

(b) Parameter-Effizientes Feintuning mit Unsloth

Abbildung 4.9. Training-Algorithmus im Remote-Backend

Die Feedback-Daten werden als Trainings- sowie als Validierungsdatensätze beim Trainingsprozess ausgewertet. Hierfür wird die Open-Source-Bibliothek *Datasets* von Hugging Face [6] (siehe Abbildung 4.10) genutzt, die die Handhabung und die Verarbeitung von großen Datensätzen bei maschinellem Lernen vereinfacht. Hugging Face [7] ist ein Unternehmen und eine frei zugängliche Plattform, die Werkzeuge und Modelle für maschinelles Lernen und natürliche Sprachverarbeitung (NLP) bereitstellt.
Im nächsten Schritt werden die Datensätze an den *SFTTrainer* übergeben, der ebenso ein Bestandteil der Hugging Face-Bibliothek [8] ist. Der SFTTrainer (kurz für Supervised Fine-Tuning Trainer) wird verwendet, um ein vortrainiertes Modell weiter auf spezifische Anwendungsfälle nach dem Prinzip des Supervised Fine-Tunings (Englisch für überwachtes Feinabstimmen) [45, vgl. Kapitel 1.6.2] zu trainieren. Dies geschieht typischerweise mit einem annotierten Datensatz, um das Modell auf bestimmte Aufgaben (in unserem Fall auf das Geoparsen) zu spezialisieren. Das heißt, dass das *dataset* (Englisch für Datensatz) verwendet wird, um das Modell anzupassen und die Modellleistung vor dem Trainingsprozess zu bewerten (siehe Abbildung 4.10 und B.2). Der Trainer bietet zusätzliche Konfigurationsoptionen (auch genannt Hyperparameter [46, 47]) die das Training effizienter gestalten, wie die Anzahl der Epochen, die Lernrate oder die Batch-Größe. Eine Epoche ist der vollständige Durchgang durch den gesamten Datensatz, während ein Durchlauf als Teilstufe innerhalb

[6] https://huggingface.co/docs/datasets/index
[7] https://huggingface.co/huggingface
[8] https://huggingface.co/docs/trl/main/en/sft_trainer

```python
EOS_TOKEN = tokenizer.eos_token
def formatting_prompts_func(data):
    inputs       = data["text"]
    outputs      = data["corrections"]
    texts = []
    for input, output in zip(inputs, outputs):
        output_str = json.dumps(output, indent=2)
        text = alpaca_prompt.format(input, output_str) + EOS_TOKEN
        texts.append(text)
    return { "text" : texts }
pass

dataset = Dataset.from_pandas(df=pd.DataFrame(request.feedback))
dataset = dataset.map(formatting_prompts_func, batched = True, load_from_cache_file=False)
```

Abbildung 4.10. Verarbeitung der Feedback-Daten zu Trainings- bwz. Validierungsdatensätze

einer Epoche eine Gruppe von Daten (Batch-Größe) betrifft. Bei jeweils einem Durchlauf, das auch als Batch-Processing (Englisch für Stapelverarbeitung [9]) bezeichnet wird, werden die Modellparameter angepasst und im nächsten Durchlauf fortgehend aktualisiert. Die Lernrate ist im Kontext von maschinellem Lernen einer der wichtigsten Hyperparameter und bestimmt, wie stark die Modellparameter als Reaktion auf den Trainingsverlust aktualisiert werden sollen. Eine zu hohe Lernrate kann zur Folge haben, dass das Modell sich zu sehr an die Trainingsdaten anpasst und ein Overfitting verursacht, wohingegen eine zu niedrige Lernrate das Modell daran hindert, aus den Daten ausreichend zu lernen und dies zu einem Underfitting führt [46]. Wie von Afaq et al. [47] beschrieben, versteht man unter Overfitting (Englisch für Überanpassung), dass das Modell zu sehr auf die Trainingsdaten feinjustiert wurde und daher keine guten Ergebnisse bei neuen, unbekannten Daten liefern kann, also schlecht generalisiert. Underfitting (Englisch für Unteranpassung) tritt auf, wenn das Modell nicht gut auf den Trainingsdaten abgestimmt ist und sowohl bei den Trainingsdaten Schwierigkeiten hat, gute Ergebnisse zu liefern, als auch bei unbekannten Daten nicht gut generalisieren kann. Der Trainingsverlust ist eine Metrik, die beschreibt, wie gut das Modell aus den bereitgestellten Daten lernt. Damit wird gemessen, wie groß der Unterschied zwischen den vorhergesagten und erwarteten Ergebnissen ist. Die Leistung des Modells wird anhand des Validierungsverlustes bewertet. Diese Metrik misst, wie gut das Modell bei einem Datensatz abschneidet, der nicht für das Training verwendet wurde [48, vgl. Kapitel 1.1] [49, vgl. Kapitel 2 A].

```python
if "finetuned" not in model_name: model_path += "-finetuned-1"
else:
    version = await get_next_version(model_name)
    model_name = f"{'-'.join(model_name.split('-')[:-1])}-{version}"
    model_path = f"models/{model_name}"
# Save model configuration and tokenizer for future finetuning
model.save_pretrained(model_path), tokenizer.save_pretrained(model_path)
# Save model as quantized GGUF-file for hosting purposes
model.save_pretrained_gguf(
    save_directory=model_path + "/gguf",
    tokenizer=tokenizer,
    quantization_method="q4_k_m"
)

# Clear GGUF-directory except quantization file
await clearGGUFDir(model_path + "/gguf")
```

(a) Speicherung der Modellkonfigurationen

```python
return {
    "message": "Finetuning process completed. Model has been saved.",
    "trainer_train_stats": trainer_train_stats[2],
    "trainer_eval_stats": trainer_eval_stats,
    "trainer_args": trainer.args
}
```

(b) Rückgabe der SFTTrainer-Argumente und der Trainingsmetriken

Abbildung 4.11. Abschluss des Trainingsprozesses

Sobald der Trainingsprozess abgeschlossen ist, werden die Modellkonfigurationen gespeichert und das Modell wird zum GPT-generierten einheitlichen Format [10] [11] (auf Englisch „GPT-Generated Unified Format“ und

[9] https://www.it-service24.com/lexikon/b/bat-dateien/
[10] https://github.com/ggerganov/ggml/blob/master/docs/gguf.md
[11] https://www.ibm.com/de-de/think/topics/gguf-versus-ggml

kurz: GGUF) konvertiert (siehe Abbildung 4.11 (a)). Das GGUF-Format wurde entwickelt, um die Effizienz bei der Nutzung von LLMs zu steigern. Der Speicherbedarf und die Ladegeschwindigkeit von LLMs werden reduziert, was bei begrenzten Hardware-Ressourcen von Vorteil ist. Die Modellparameter und -konfigurationen werden in einer Einzeldatei komprimiert. Zudem ist GGUF optimal für das Hosting von LLMs geeignet, insbesondere in Szenarien, die schnelle Antworten und eine skalierbare Bereitstellung erfordern. Dadurch können datenintensive Aufgaben wie Georeferenzierungsaufgaben besser bewältigt werden. Anschließend werden die Metriken Trainings- und Validierungsverlust mitsamt der verwendeten Hyperparameter und einer Bestätigung des erfolgreichen Trainingsprozesses an die POST-Anfrage rückgemeldet. Im Backend werden diese Daten mit MLflow protokolliert (siehe Abbildungen B.3 und C.1).

4.3 Architektur

Ingesamt gliedert sich die Implementierung des Systems in drei Hauptkomponenten: das Frontend, das eine benutzerfreundliche Schnittstelle für die Interaktion mit den annotierten Daten bietet, das Backend, welches API-Schnittstellen zur Verarbeitung von Daten und dem Geoparsen von Webdaten bietet, und das Remote-Backend, das auf Grundlage von Feedback-Daten das im Provider spezifizierte LLM evaluiert und trainiert.

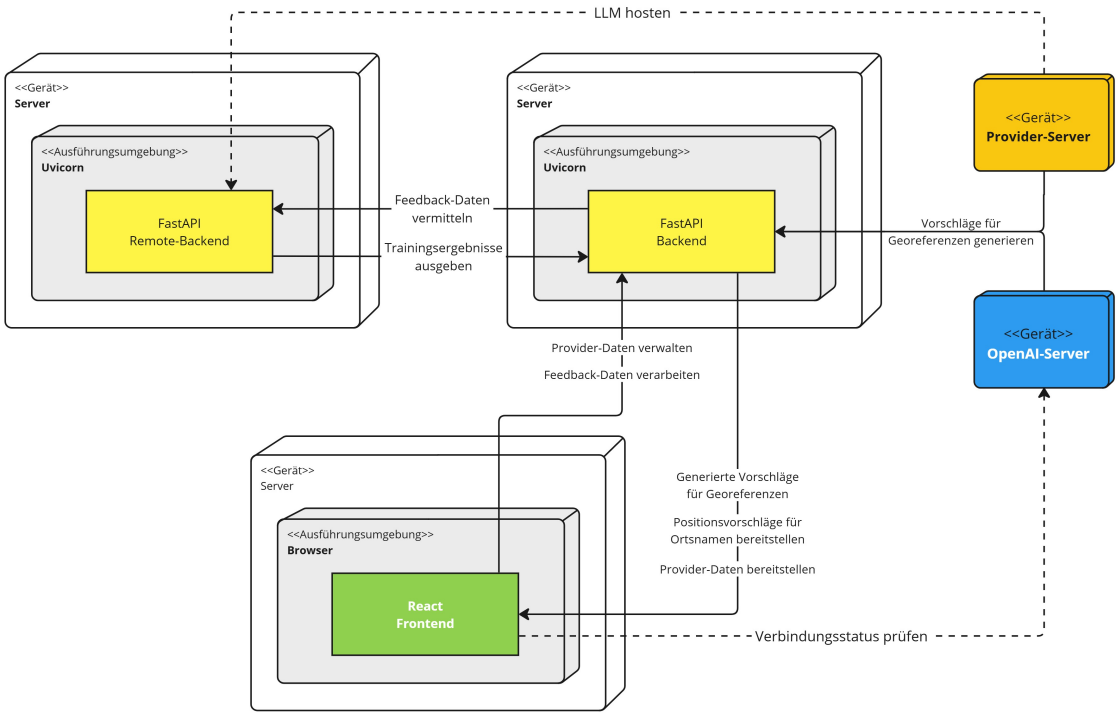


Abbildung 4.12. Verteilungsansicht des GeoSense Annotators

Das Verteilungsdiagramm (siehe Abbildung 4.12) präsentiert die Architektur des GeoSense Annotators und illustriert die Interaktionen zwischen dem Frontend, dem Backend und dem Remote-Backend, wie dem Provider-Server und dem OpenAI-Server. Die Provider-Management-Komponente im Frontend (siehe Abbildung 4.13 kommuniziert über eine API-Schnittstelle mit dem Backend und tauscht Daten wie Provider-Daten oder

Feedback-Daten aus oder bezieht Vorschläge für Georeferenzen oder Positionen für gekennzeichnete Ortsnamen in der Text- und Annotationskomponente.

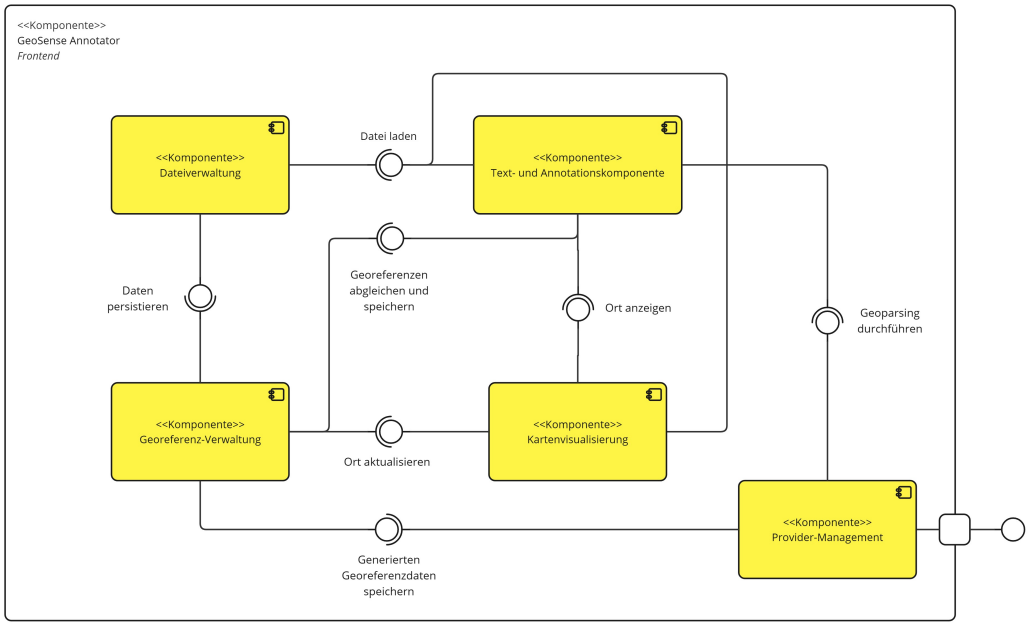


Abbildung 4.13. Komponentendiagramm vom Frontend des GeoSense Annotators

Das Backend, das als zentrale Verarbeitungskomponente dient, speichert und verwaltet Provider-Daten, holt die Vorschläge für Georeferenzen vom Provider- oder OpenAI-Server ein und bereitet die Feedback-Daten in der Active-Learning-Komponente für den Validierungs- und Trainingsprozess im Remote-Backend vor. Im Anschluss werden die Trainingsergebnisse vom Remote-Backend bezogen und protokolliert.

Kapitel 5

Evaluation

In diesem Abschnitt wird nicht die gesamte Funktionalität des GeoSense Annotators evaluiert, sondern ausschließlich die Active-Learning-Komponente des Annotationstools, weil die Arbeit darauf abzielt, die Genauigkeit und Verbesserung des eingesetzten Modells durch schrittweises Training zu untersuchen, basierend auf den generierten und ggfs. korrigierten Georeferenzen und den berechneten Bewertungsmetriken. Die Ergebnisse spiegeln daher lediglich die Leistung dieser spezifischen Komponente wider und nicht die des gesamten Systems.

5.1 Ansatz

Um die Präzision und die Effizienz der Georeferenzierung zu bewerten und das LLM effektiv trainieren zu können, wurde ein umfangreiches, mit Georeferenzen annotiertes Korpus vorausgesetzt. Das Korpus sollte die durch das LLM generierten Georeferenzen als auch die durch einen Menschen korrigierten Georeferenzen beinhalten. Einen umfangreichen Datensatz zu annotieren, nimmt jedoch erheblich viel Zeit in Anspruch. Statt Georeferenzen mithilfe des GeoAnnotators zu annotieren und zu korrigieren, wurde das Korpus von Liebermann et al. [50], der Local Globus Corpus (LGL) [1] verwendet. Wie von Sheikh et al. [3, vgl. Kapitel 5.1] beschrieben, besteht das LGL-Korpus aus 557 Nachrichtenartikeln, die jeweils mit Standortinformationen, einschließlich geographischer Koordinaten, von Experten annotiert wurden. Die Nutzung dieses Datensatzes bot eine zuverlässige Basis für Tests und erlaubte es, sich auf das Training und Evaluieren des Modells zu konzentrieren. Außerdem konnte der zeitintensive Schritt der manuellen Annotation übersprungen werden. Im Rahmen der schrittweisen Testmethode wurde das Korpus jeweils in 6 Datensätze mit je 100 Daten, abgesehen von einem Datensatz mit 57 Daten, für das Training und die Evaluation des Modells aufgeteilt. Die Datensätze wurden in chronologischer Reihenfolge mit dem stufenweise trainierten Modell verarbeitet. Zunächst wurde der erste Datensatz mit dem Basis-Modell, *Llama-3.1-8B-Instruct*, verarbeitet, um die initialen Georeferenzen zu generieren. Vor dem Trainingsprozess wurden die generierten Annotationen mit den bestehenden Annotationen verglichen und die Bewertungsmetriken *Precision*, *Recall*, *F1-Score* und das *Accuracy at k* ($A@k$) berechnet. Daraufhin wurde das Modell mit den bestehenden Annotationen, das als menschliches Feedback verstanden werden kann, feinjustiert und für den nächsten Schritt vorbereitet. Allerdings wurde die Modellleistung während dem Trainingsprozess erst mit dem nächsten Datensatz validiert, um so einen umfangreichen Trainings- als auch Validierungsdatensatz bereitzustellen. Beispielsweise wurde das erste feinjustierte Modell nicht mit dem ersten Trainingsdatensatz validiert, sondern mit dem zweiten Datensatz,

[1] https://github.com/milangritta/Pragmatic-Guide-to-Geoparsing-Evaluation/blob/master/data/Corpora/lgl.xml

der ebenso als Trainingsdatensatz für den zweiten Trainingsprozess diente. Die restlichen Datensätze wurden nach diesem Schema iterativ weiterverarbeitet. Nach jedem Durchlauf wurden außerdem die Metriken *Trainingsverlust* und *Validierungsverlust* mitberücksichtigt.

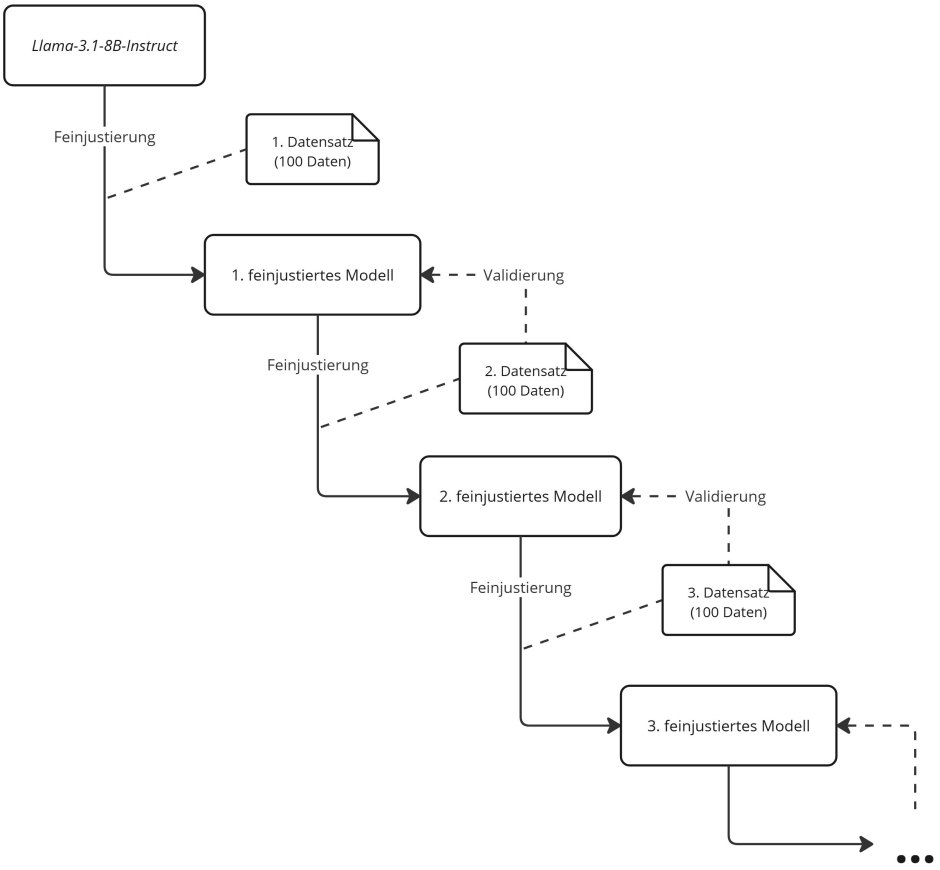


Abbildung 5.1. Trainings- und Validierungsprozess

Das Training erfolgte in zwei Epochen und einer Lernrate von 2e-4. Es wurde eine Batch-Größe von 2 gewählt. Die Wahl einer bestimmten Zahl an Epochen ist nicht fest vorgegeben und variiert von Datensatz zu Datensatz [47, vgl. Kapitel 7]. Um sicherzustellen, dass eine moderate Anzahl gewählt wurde, müssen die Metriken Trainings- und Validierungsverlust beobachtet und ausgewertet werden. Sobald ein Over- oder Underfitting festgestellt wird, kann davon ausgegangen werden, dass entweder die Epochenzahl falsch eingeschätzt wurde oder die Lernrate unpassend ist. Die Wahl einer Lernrate von 2e-4 und einer Batch-Größe von 2 wurde bewusst getroffen, um eine Balance zwischen schneller Konvergenz und Stabilität im Trainingsprozess sicherzustellen. Die gewählte Lernrate stellt sicher, dass das Modell in moderatem Tempo lernt und das Risiko eines Overfitting bei einer geringen Anzahl an Epochen minimiert wird, wohingegen die gewählte Batch-Größe mehrere Gründe hat. Zum einen ist die Wahl einer kleinen Batch-Größe vorteilhaft bei einer begrenzten Rechenleistung [51]. Die Berechnung des Trainings- als auch Validierungsverlustes benötigt bei einer kleinen Batch-Größe weniger GPU-Speicher. Zum anderen beeinflusst die Wahl der Batch-Größe die Lernleistung des Modells. Keskar et al. [52] argumentieren, dass eine große Batch-Größe das Erreichen eines minimalen Trainingsverlustes erschwert. Im Vergleich vereinfachen kleinere Batch-Größen das Erreichen eines Minimums.

Des Weiteren war vorgesehen, die Geoparser *CamCoder*, *CLAVIN* und *Mordecai* zu verwenden, um das LGL-Korpus vollständig zu verarbeiten und jeweils Georeferenzen zu generieren. Das DLR hat einen mit den genannten Geoparsern bereits annotierten Datensatz bereitgestellt, sodass der Schritt der vollständigen Verarbeitung des LGL-Korpus durch die Geoparser übersprungen werden konnte. Aus dem Datensatz wurden die Bewertungsmetriken F1-Score und A@k errechnet, die im Anschluss mit dem Mittelwert der Bewertungsmetriken des trainierten Modells verglichen wurden. Daran konnte festgemacht werden, wie gut das LLM im Vergleich zu den herkömmlichen Geoparsern abgeschnitten hat.

5.2 Ergebnisse und Diskussion

Aus der Tabelle 5.1 können die Entwicklungen der Bewertungsmetriken der jeweiligen feinjustierten Modelle entnommen werden. Hierbei werden die Metriken **Precision**, **Recall**, **F1-Score** sowie die **Accuracy at k** ($A@k$) für die Radien 10 km und 161 km betrachtet. Die Bewertungsmetriken dienen als zuverlässige Grundlage zur Beurteilung der Präzision und Effizienz der Georeferenzierung. Um einen Gesamteindruck der Modellleistung zu erhalten, reicht die Metrik F1-Score aus, die die Balance zwischen Precision und Recall angibt. Anhand der Metrik Precision wird geprüft, wie viele der von dem Modell als Ortsnamen identifizierten Einträge tatsächlich korrekt sind, wohingegen die Metrik Recall angibt, wie viele der tatsächlichen Ortsnamen korrekt erkannt wurden. Dabei hilft die Metrik A@k, die Genauigkeit der Positionen zu bewerten.

Feinjustiertes Modell	Precision	Recall	F1-Score	A@k - 10	A@k - 161
Basis-Modell	65.47	70.69	67.99	39.64	85.09
1.	73.47	64.09	68.47	34.44	89.63
2.	61.32	80.46	69.59	39.12	87.70
3.	70.45	83.50	76.42	33.74	86.32
4.	62.00	80.48	70.04	45.12	90.57
5.	31.65	86.67	46.37	34.07	90.11

Tabelle 5.1. Bewertungsmetriken der Modelle

Die Werte beim Basismodell deuten darauf hin, dass das Modell bereits eine fundierte Grundfähigkeit zur Georeferenzierung besitzt, jedoch Verbesserungspotenzial aufweist. Der F1-Score von **67.66%** zeigt, dass das Modell ohne vorherige Feinjustierung in der Lage ist, eine relativ gute Balance zwischen Precision (**65.47%**) und Recall (**70.69%**) zu erreichen, was darauf hinweist, dass es eine adäquate Anzahl relevanter Ortsnamen korrekt identifizieren kann. Die hohe A@k für 161 km (**85,09%**) deutet darauf hin, dass das Modell besonders in einem größeren Toleranzbereich gute Ergebnisse liefert. Allerdings sinkt die Genauigkeit bei einem kleineren Toleranzbereich (10 km) auf **39,64%**, was darauf hindeutet, dass das Modell Schwierigkeiten hat, Ortsnamen präzise innerhalb eines kleinen Radius zuzuordnen. Diese Werte legen nahe, dass das Modell bereits ohne Training brauchbare Ergebnisse liefert, jedoch von einer gezielten Anpassung und Feinjustierung profitieren könnte, um die Genauigkeit in kleineren Toleranzbereichen und die Balance zwischen Precision und Recall weiter zu verbessern.

Nach der ersten Feinjustierung zeigt das Modell einen leichten Anstieg der Precision auf **73.47%**, was bedeutet, dass es weniger fehlerhafte Ortsnamen extrahiert. Zugleich sinkt jedoch der Recall auf **64.09%**, was darauf hindeutet, dass weniger relevante Ortsnamen erkannt wurden. Dies führt zu einem leichten Rückgang

im F1-Score (**68,47%**). Obwohl die Genauigkeit der Positionen für den Toleranzbereich von 10 km leicht gesunken ist, liefert das Modell für den Radius 161 km bessere Ergebnisse.

Das zweite feinjustierte Modell erreicht eine deutliche Steigerung des Recall auf **80.46%**, während die Precision leicht absinkt. Dies zeigt, dass das Modell nun mehr relevante Georeferenzen erkennt, allerdings auf Kosten einer leicht erhöhten Fehlerquote. Der F1-Score verbessert sich auf **69.59%**, was eine gute Balance zwischen Precision und Recall widerspiegelt. Die A@k bei 10 km und 161 km verbessert sich ebenfalls leicht.

Die feinjustierten Modelle 3 und 4 erreichen einen merklichen Anstieg sowohl bei Precision als auch bei Recall, was zu einem höheren F1-Score führt (**76.42%** bzw. **70.04%**). Insbesondere das vierte feinjustierte Modell zeigt die bisher besten Ergebnisse im Hinblick auf die A@k-Metriken, was darauf hindeutet, dass es die Positionen präziser zuordnen kann, besonders im kleineren Toleranzbereich (**45,12%** bei A@k-10).

Das fünfte Modell zeigt einen deutlichen Abfall in der Precision (**31.65%**), während der Recall auf **86.67%** ansteigt. Dies könnte darauf hinweisen, dass das Modell zwar viele relevante Ortsnamen erkennt, dabei jedoch zahlreiche falsche Zuordnungen macht. Der F1-Score sinkt hier auf **46.87%**, was auf eine Verschlechterung der Gesamtleistung hindeutet. Diese Verschlechterung zeigt sich auch in den A@k-Metriken, insbesondere bei A@k-10.

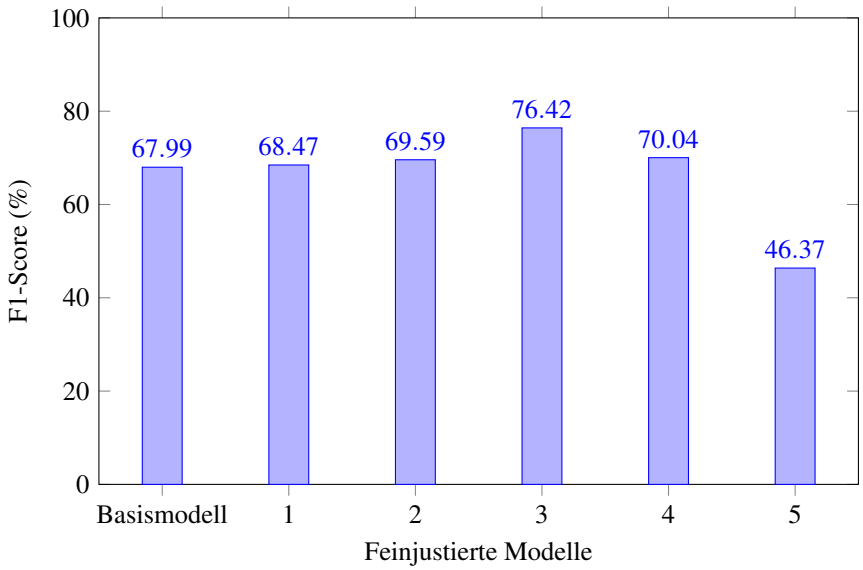


Abbildung 5.2. F1-Score der feinjustierten Modelle

Zusammenfassend lässt sich sagen, dass das Basismodell bereits präzise Ergebnisse liefert und mit zunehmendem Training eine bessere Modellleistung vorweist. Dies lässt sich an der anfänglichen positiven Entwicklung des F1-Score (siehe Abbildung 5.2) nachweisen. Jedoch tritt beim letzten Bewertungsprozess eine Verschlechterung der Precision auf. Dies ist jedoch darauf zurückzuführen, dass im letzten Durchlauf halb so viele Daten verarbeitet wurden als bei den vorherigen Durchläufen. Im letzten Schritt wurden lediglich 57 Daten statt 100 Daten verwendet. Eine kleinere Datenmenge führt zu einer höheren statistischen Unsicherheit und Varianz in den Leistungsmetriken. Denn mit weniger Daten ist es umso wahrscheinlicher, dass zufällige Abweichun-

gen das Ergebnis stärker beeinflussen, da es weniger Daten gibt, die diese Schwankungen „ausgleichen“. Außerdem führt ein geringerer Anteil an wichtigen Variationen und seltenen Fällen im Datensatz zu einer schlechteren Generalisierung. Das Modell kann somit Schwierigkeiten haben, komplexe oder ungewöhnliche Georeferenzierungsaufgaben korrekt zu bewältigen. Ferner ist es wichtig, ein Over- oder Underfitting zu vermeiden, um eine kohärente Generalisierung des Modells beim Training zu ermöglichen.

Trainingsiteration	Trainingsverlust	Validierungsverlust
1	0.825	1.481
2	0.800	1.217
3	0.718	1.229
4	0.755	1.133
5	0.648	1.339

Tabelle 5.2. Trainings- und Validierungsverlust pro Trainingsiteration

Hierfür betrachten wir die Entwicklung des Trainings- und Validierungsverlustes [47, vgl. Kapitel 3 B.] im Laufe der Trainingsiteration in Tabelle 5.2. Es ist erkennbar, dass der Trainingsverlust fortlaufend abnimmt, was auf die effektive Anpassung des Modells an die Trainingsdaten hindeutet. Zugleich zeigt der Validierungsverlust eine ähnliche Tendenz und stabilisiert sich. Diese Entwicklung verdeutlicht, dass das Modell sich nicht zu sehr an die Trainingsdaten anpasst und eine angemessene Generalisierungsfähigkeit besitzt. Somit ist die Gefahr eines Overfittings relativ gering. Ein leichter Anstieg des Validierungsverlustes ist jedoch erkennbar, der in potentiell weiteren Trainingsiterationen hätte beobachtet werden müssen.

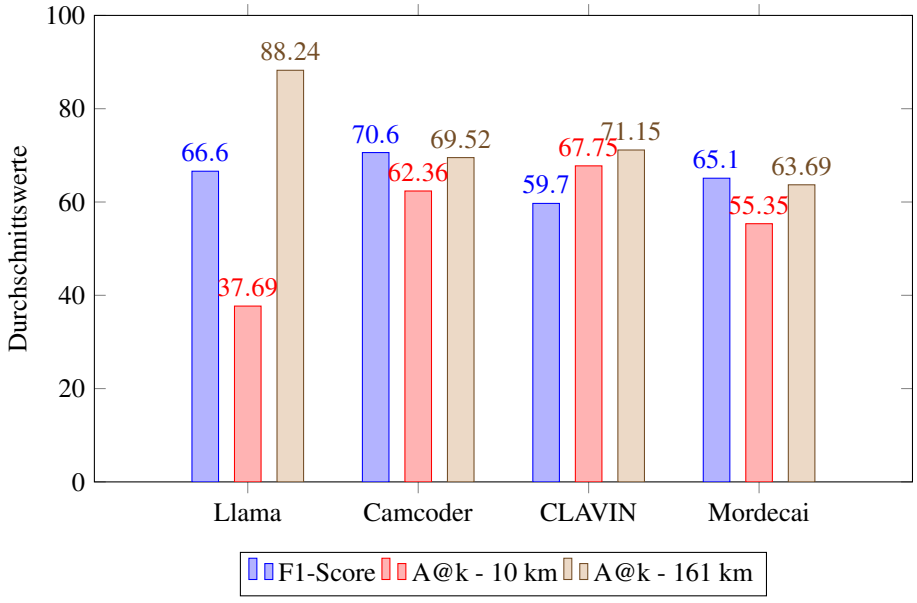


Abbildung 5.3. Vergleich der durchschnittlichen F1-Scores und A@k-Werte

Die Abbildung 5.3 zeigt den Vergleich der durchschnittlichen F1-Scores sowie der A@k-Werte (für 10 km und 161 km) zwischen dem trainierten Llama-Modell und den Geoparsern *Camcoder*, *CLAVIN* und *Mordecai*. Das Modell erreicht einen F1-Score von **66,6%** und liegt damit hinter Camcoder (**70,6%**), , aber vor CLAVIN (**59.7%**) und Mordecai (**65.1%**). Bei der A@k-Metrik für 10 km schneidet das Modell mit einem

Wert von **37,69%** am schwächsten ab, während die Geoparser höhere Präzisionswerte zeigen, insbesondere CLAVIN mit **67,75%**. Für die A@k-Metrik bei 161 km übertrifft das Modell jedoch die Geoparser deutlich mit einem Wert von **88,24%**. Diese Ergebnisse deuten darauf hin, dass das trainierte Modell in Bezug auf die geographische Genauigkeit bei größerem Toleranzbereich (161 km) überlegen ist, jedoch in engeren geographischen Bereichen (10 km) hinter den spezialisierten Geoparsern zurückbleibt.

Dies zeigt, dass das Modell in breiteren geographischen Kontexten konkurrenzfähig ist, aber weiterhin optimiert werden müsste, um mit spezialisierten Geoparsern mitzuhalten.

Kapitel 6

Fazit und Ausblick

Das Hauptziel dieser Arbeit war es, die effektive Integration von LLMs bei der semi-automatischen Annotation von Webdaten mit Georeferenzierung in dem GeoSense Annotator zu untersuchen. Die zentrale Arbeitshypothese lautete, dass der Einsatz von LLMs die Präzision der Georeferenzierung erheblich steigern würde, da LLMs kontextuelle Informationen besser nutzen können und somit den manuellen Nachbearbeitungsaufwand reduzieren. Durch die iterative Verbesserung der eingesetzten LLMs sollte unter anderem die Qualität der Georeferenzierung verbessert werden. Die Ergebnisse der Arbeit zeigen, dass das eingesetzte LLM (*Llama-3.1-8B-Instruct*) in der Lage war, Georeferenzen aus unstrukturierten Texten zuverlässig zu extrahieren und die Genauigkeit der Annotationen durch das Training nach dem Prinzip des Active Learnings signifikant zu erhöhen. Des Weiteren konnte nachgewiesen werden, dass auch bei einer geringen Zahl an Trainingsdatensätzen das Training des Modells effektiv durchgeführt werden konnte, ohne dass ein anfängliches Over- oder Underfitting vorgefunden wurde. Obwohl der schrittweise Trainingsprozess eine fortlaufende Verbesserung der Modellleistung zeigt, sind möglicherweise Schwankungen in den Metriken durch die chronologische Verarbeitung der Datensätze verursacht worden. Diese Schwankungen könnten auf die spezifischen Eigenschaften der chronologischen Reihenfolge zurückzuführen sein, beispielsweise auf die Unterschiede in der Datenkomplexität zwischen den Datensätzen. Eine mögliche Optimierung könnte darin bestehen, die Datensätze vor der Aufteilung zufällig zu mischen. Dadurch würde die Validierungs- und Trainingsreihenfolge von möglichen Abhängigkeiten in der Datenstruktur entkoppelt werden. Es ist anzunehmen, dass sich durch eine solche Zufallsmischung und die Wiederholung des Experiments ein stabilerer positiver Trend in den Validierungsmetriken ergibt, insbesondere wenn die Ergebnisse gemittelt werden. Aufgrund der begrenzten Zeit konnten jedoch keine weiteren Experimente mit zufälliger Reihenfolge durchgeführt werden. Dies könnte jedoch eine vielversprechende Richtung für zukünftige Arbeiten darstellen, um die Robustheit und Fähigkeit des Modells, Georeferenzen aus Webdaten zu extrahieren, weiter zu verbessern. Diese Ergebnisse sind bedeutend, da sie zeigen, dass der Einsatz von LLMs in Annotationstools wie dem GeoSense Annotator eine vielversprechende Alternative mit erheblichem Verbesserungspotential (wie in der Arbeit von Bhandari et al. [10] erläutert) zu bestehenden Georeferenzierungsmethoden wie CamCoder, CLAVIN oder Mordecai darstellen und der GeoSense Annotator zur Verbesserung der Genauigkeit in der geographischen Informationsverarbeitung beitragen könnte.

In zukünftigen Arbeiten bieten sich verschiedene Optimierungs- und Erweiterungsmöglichkeiten an, um den GeoSense Annotator weiter zu verbessern und ihre Einsatzmöglichkeiten auszubauen:

- **Verbesserungspotential:**
  - **Erweiterung des Korpus**: Durch zusätzliche Datenannotation mit dem GeoSense Annotator oder die Zusammenführung von verschiedenen bestehenden Korpora könnte ein vielfältigeres Korpus erstellt werden, womit die Robustheit und Genauigkeit der Modelle verbessert werden würde.
  - **Optimierung des LLaMA-Modells**: Durch gezielte Feinabstimmung und zusätzliches Training könnte die Präzision und Effizienz des *LLaMA 3.1 8B Instruct*-Modells verbessert werden, insbesondere im Hinblick auf die Positionsgenauigkeit innerhalb eines Radius von 10 km.
  - **Benutzerschnittstelle**: Auf der Provider-Seite des GeoSense Annotators könnte ein optionales Button angeboten werden, um das Training eines Modell manuell zu starten, auch wenn der vordefinierte Schwellenwert nicht erreicht wurde. Darüber hinaus könnten optionale Felder integriert werden, um die Bewertungsmetriken und Trainingsergebnisse darzustellen und die Hyperparameter wie die Lernrate, die Epochenzahl oder die Batch-Größe entsprechend den Trainingsergebnissen anzupassen. Diese Flexibilität würde den Nutzern eine bessere Kontrolle über die Trainingsprozesse gewährleisten.
- **Zukünftige Forschung:**
  - **Integration neuer Technologien**: Die Erforschung und Integration neuerer, leistungsstärkerer LLMs von LLaMA, OpenAI oder Google, die mit umfangreicheren und zahlreichen Datensätzen trainiert wurden, könnte die Automatisierung und Präzision der Georeferenzierung weiter vorantreiben und unerforschte Potentiale für die Georeferenzierung aufweisen.
  - **Interdisziplinäre Ansätze**: Die Kombination der Georeferenzierung mit GeoSense Annotator und der Umweltwissenschaften oder Stadtplanung könnte wertvolle neue Perspektiven eröffnen. So könnten bessere Entscheidungen in Bereichen wie Verkehrsplanung, Grünflächenentwicklung oder Ressourcennutzung durch die Nutzung des Tools auf Grundlage von georeferenzierten Daten aus Bürgerbefragungen, Berichten und soziale Medien getroffen werden.
  - **Langzeitstudien**: Langzeitstudien in Bezug auf den GeoSense Annotator wären notwendig, um die Nachhaltigkeit und Effizienz des entwickelten Ansatzes über längere Zeiträume zu bewerten.
- **Praktische Anwendungen:**
  - **Geoinformationssysteme (GIS)**: Der GeoSense Annotator könnte in GIS zur Verbesserung der Standortplanung und Umweltforschung integriert werden.
  - **Katastrophenmanagement**: Eine präzisere Georeferenzierung durch die kontinuierliche Verbesserung der eingesetzten LLMs könnte eine schnellere und genauere Reaktion auf Naturkatastrophen ermöglichen.
  - **Öffentliche Stadtverwaltung**: Die öffentliche Stadtverwaltung, wie zum Beispiel das Bauamt, könnte durch den GeoSense Annotator bei der Verwaltung und Analyse geographischer Daten unterstützt werden. Beispielsweise können georeferenzierte Daten bessere Hinweise liefern, wo infrastrukturelle Änderungen am dringendsten erforderlich sind.

Dieser Ausblick unterstreicht die zukünftigen Potentiale zur technischen Verbesserung, forschungsmethodischen Weiterentwicklung und praktischen Anwendung des GeoSense Annotators.

Literaturverzeichnis

[1] Sheikh Mastura Farzana and Tobias Hecking. Geoparsing at Web-scale - Challenges and Opportunities. In *GeoExT 2023: First International Workshop on Geographic Information Extraction from Texts at ECIR 2023*. German Aerospace Center (DLR), Institute for Software Technology, 2023. https://ceur-ws.org/Vol-3385/paper6.pdf.

[2] Michael Granitzer, Stefan Voigt, Noor Afshan Fathima, Martin Golasowski, Christian Guetl, Tobias Hecking, Gijs Hendriksen, Djoerd Hiemstra, Jan Martinovič, Jelena Mitrović, Izidor Mlakar, Stavros Moiras, Alexander Nussbaumer, Per Öster, Martin Potthast, Marjana Senčar Srdič, Sharikadze Megi, Kateřina Slaninová, Benno Stein, Arjen P. de Vries, Vít Vondrák, Andreas Wagner, and Saber Zerhoudi. Impact and development of an Open Web Index for open web search. *Journal of the Association for Information Science and Technology*, 75(5):512–520, 2024. https://doi.org/10.1002/asi.24818,.

[3] Sheikh Mastura Farzana and Tobias Hecking. Towards a Scalable Geoparsing Approach for the Web. In *GeoExT 2024: Second International Workshop on Geographic Information Extraction from Texts at ECIR 2024*. German Aerospace Center (DLR), Institute for Software Technology, 2024. https://ceur-ws.org/Vol-3683/paper4.pdf.

[4] Milan Gritta, Mohammad Taher Pilehvar, Nut Limsopatham, and Nigel Collier. What's missing in geographical parsing? *Language Resources & Evaluation*, 52(2):603–623, 2018. https://doi.org/10.1007/s10579-017-9385-8.

[5] Rohin Manvi, Samar Khanna, Gengchen Mai, Marshall Burke, David Lobell, and Stefano Ermon. Geollm: Extracting geospatial knowledge from large language models, 2024.

[6] Leonardo Nizzoli, Marco Avvenuti, Maurizio Tesconi, and Stefano Cresci. Geo-semantic-parsing: AI-powered geoparsing by traversing semantic knowledge graphs. *Decision Support Systems*, 136:113346, 2020. https://www.sciencedirect.com/science/article/pii/S0167923620301019.

[7] Daniel Ferrés Domènech and Horacio Rodríguez Hontoria. Georeferencing textual annotations and tagsets with geographical knowledge and language models. 2011. https://api.semanticscholar.org/CorpusID:16196858.

[8] Xuke Hu, Zhiyong Zhou, Hao Li, Yingjie Hu, Fuqiang Gu, Jens Kersten, Hongchao Fan, and Friederike Klan. Location Reference Recognition from Texts: A Survey and Comparison. *ACM Computing Surveys*, 56(5):112, 2023. https://dl.acm.org/doi/pdf/10.1145/3625819.

[9] C. J. Rupp, Paul Rayson, Ian N. Gregory, Andrew Hardie, Amelia Joulain, and Daniel Hartmann. Dealing with heterogeneous big data when geoparsing historical corpora. *2014 IEEE International Confe-*

rence on Big Data (Big Data), pages 80–83, 2014. https://api.semanticscholar.org/CorpusID:16427335.

[10] Prabin Bhandari, Antonios Anastasopoulos, and Dieter Pfoser. Are Large Language Models Geospatially Knowledgeable? In *The 31st ACM International Conference on Advances in Geographic Information Systems (SIGSPATIAL '23)*. ACM, 2023. https://dl.acm.org/doi/pdf/10.1145/3589132.3625625.

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, May 2019. http://arxiv.org/abs/1810.04805.

[12] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners, 2020. https://arxiv.org/pdf/2005.14165.

[13] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and Efficient Foundation Language Models, February 2023. https://arxiv.org/pdf/2302.13971.

[14] Andreas Hackeloeer, Klaas Klasing, Jukka M. Krisp, and Liqiu Meng. Georeferencing: a review of methods and applications. *Annals of GIS*, 20(1):61–69, January 2014. Publisher: Taylor & Francis, https://doi.org/10.1080/19475683.2013.868826,.

[15] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Z. Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jianyun Nie, and Ji rong Wen. A Survey of Large Language Models. *ArXiv*, abs/2303.18223, 2023. https://api.semanticscholar.org/CorpusID:257900969.

[16] Muhammad Usman Hadi, al tashi, Rizwan Qureshi, Abbas Shah, Muhammad Irfan, Anas Zafar, Muhammad Bilal Shaikh, Naveed Akhtar, Jia Wu, Seyedali Mirjalili, Qasem Al-Tashi, Amgad Muneer, Mohammed Ali Al-garadi, Gru Cnn, and T5 RoBERTa. Large language models: A comprehensive survey of its applications, challenges, limitations, and future prospects. December 2023. https://api.semanticscholar.org/CorpusID:266378240.

[17] Jeff Heaton. Ian Goodfellow, Yoshua Bengio, and Aaron Courville: Deep learning. *Genetic Programming and Evolvable Machines*, 19(1):305–307, June 2018. https://doi.org/10.1007/s10710-017-9314-z.

[18] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6000–6010, Red Hook, NY, USA, 2017. Curran Associates Inc.

[19] Navin C. Sabharwal and Amit Agrawal. BERT Model Applications: Other Tasks. 2021. https://api.semanticscholar.org/CorpusID:234321040.

[20] Daniel W. Goldberg. *Geocoding*, pages 1–12. John Wiley & Sons, Ltd, 2017. https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118786352.wbieg1051.

[21] Milan Gritta, Mohammad Taher Pilehvar, and Nigel Collier. Which Melbourne? Augmenting Geocoding with Maps. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Long Papers)*, pages 1285–1296. Association for Computational Linguistics, 2018. https://aclanthology.org/P18-1119.pdf.

[22] Jimin Wang, Yingjie Hu, and Kenneth Joseph. Neurotpr: A neuro-net toponym recognition model for extracting locations from social media messages. *Transactions in GIS*, 24:719–735, 2020. https://doi.org/10.1111/tgis.12627.

[23] Anirudha Ghosh, Abu Sufian, Farhana Sultana, Amlan Chakrabarti, and Debashis De. *Fundamental Concepts of Convolutional Neural Network*, pages 519–567. Springer International Publishing, Cham, 2020. https://doi.org/10.1007/978-3-030-32644-9_36.

[24] Jimin Wang and Yingjie Hu. Enhancing spatial and textual analysis with eupeg: An extensible and unified platform for evaluating geoparsers. *Transactions in GIS*, 23(6):1393–1419, October 2019. http://dx.doi.org/10.1111/tgis.12579.

[25] Andrew Halterman. Mordecai: Full Text Geoparsing and Event Geocoding. *The Journal of Open Source Software*, 2(9), 2017. https://doi.org/10.21105/joss.00091.

[26] Andrew Halterman. Mordecai 3: A Neural Geoparser and Event Geocoder. *arXiv preprint arXiv:2303.13675*, 2023. https://arxiv.org/pdf/2303.13675.

[27] Tomas Mikolov, Greg Corrado, Kai Chen, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. *arXiv preprint arXiv:1301.3781*, 2013. https://arxiv.org/pdf/1301.3781.

[28] Morteza Karimzadeh and Alan M. MacEachren. GeoAnnotator: A Collaborative Semi-Automatic Platform for Constructing Geo-Annotated Text Corpora. *ISPRS Int. J. Geo Inf.*, 8(4):161, 2019. https://api.semanticscholar.org/CorpusID:108373088.

[29] Yiting Ju, Benjamin Adams, Krzysztof Janowicz, Yingjie Hu, Bo Yan, and Grant McKenzie. Things and strings: Improving place name disambiguation from short texts by combining entity Co-occurrence with topic modeling. In *Lecture Notes in Computer Science*, volume 10024 of *LNAI*, pages 353–367. Springer, 2016.

[30] Kalev H. Leetaru. Fulltext Geocoding Versus Spatial Metadata for Large Text Archives: Towards a Geographically Enriched Wikipedia. *D-Lib Magazine*, 18(5), 2012.

[31] Jochen Lothar Leidner. *Toponym Resolution in Text: Annotation, Evaluation and Applications of Spatial Grounding of Place Names*. Doctor of philosophy, University of Edinburgh, Edinburgh, UK, 2007. http://hdl.handle.net/1842/1849.

[32] Jochen L. Leidner and Luca Jung. Tame ii: A modern geographic text annotation tool. In Maryam Lotfian and Luigi Libero Lucio Starace, editors, *Web and Wireless Geographical Information Systems*, pages 95–104, Cham, 2024. Springer Nature Switzerland.

[33] Beatrice Alex, Kate Byrne, Claire Grover, and Richard Tobin. A Web-based Geo-resolution Annotation and Evaluation Tool. In *LAW VIII-The 8th Linguistic Annotation Workshop*, pages 59–63, Dublin, Ireland, August 23-24 2014. Association for Computational Linguistics. https://aclanthology.org/W14-4908.pdf.

[34] Greg DeLozier, Benjamin Wing, Jason Baldridge, and Scott Nesbit. Creating a Novel Geolocation Corpus from Historical Texts. In *Proceedings of the 10th Linguistic Annotation Workshop Held in Conjunction with ACL 2016 (LAW-X 2016)*. Association for Computational Linguistics, August 11 2016. https://aclanthology.org/W16-1721.

[35] Morteza Karimzadeh, Scott Pezanowski, Alan M. MacEachren, and Jan O. Wallgrün. Geotxt: A scalable geoparsing system for unstructured text geolocation. *Transactions in GIS*, 23(1):118–136, 2019. https://doi.org/10.1111/tgis.12510.

[36] Winfried Höhn, Hans-Günter Schmidt, and Hendrik Schöneberg. Semiautomatic recognition and georeferencing of places in early maps. In *Proceedings of the 13th ACM/IEEE-CS Joint Conference on Digital Libraries*, JCDL '13, page 335–338, New York, NY, USA, 2013. Association for Computing Machinery. https://doi.org/10.1145/2467696.2467734.

[37] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural Machine Translation of Rare Words with Subword Units. *arXiv preprint arXiv:1508.07909*, 2016. https://arxiv.org/pdf/1508.07909.

[38] Burr Settles. Active Learning Literature Survey. Technical report, University of Wisconsin-Madison, 2009. Technical Report, Computer Sciences Department, https://axon.cs.byu.edu/~martinez/classes/778/Papers/settles.activelearning.pdf.

[39] Sergio A. Alvarez. An exact analytical relation among recall , precision , and classification accuracy in information retrieval. 2002. https://api.semanticscholar.org/CorpusID:7632639.

[40] John Wieczorek, Qinghua Guo, and Robert Hijmans. The point-radius method for georeferencing locality descriptions and calculating associated uncertainty. *International Journal of Geographical Information Science*, 18(8):745–767, 2004. https://doi.org/10.1080/13658810412331280211.

[41] P Dauni, M D Firdaus, R Asfariani, M I N Saputra, A A Hidayat, and W B Zulfikar. Implementation of Haversine formula for school location tracking. *Journal of physics. Conference series*, 1402(7):77028–, 2019. https://www.proquest.com/scholarly-journals/implementation-haversine-formula-school-location/docview/2568300124/se-2,.

[42] Ankush Singal. Unleashing the Power of Unsloth and QLora: Redefining Language Model Fine-Tuning. Hugging-Face-Blog, https://huggingface.co/blog/Andyrasika/finetune-unsloth-qlora, Zugriff am 19. September 2024.

[43] Charith Chandra Sai Balne, Sreyoshi Bhaduri, Tamoghna Roy, Vinija Jain, and Aman Chadha. Parameter Efficient Fine Tuning: A Comprehensive Analysis Across Applications. *ArXiv*, abs/2404.13506, 2024. https://api.semanticscholar.org/CorpusID:269293949.

[44] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. QLoRA: Efficient Finetuning of Quantized LLMs. *arXiv preprint*, 2023. https://arxiv.org/pdf/2305.14314,.

[45] Venkatesh Balavadhani Parthasarathy, Ahtsham Zafar, Aafaq Khan, and Arsalan Shahid. The Ultimate Guide to Fine-Tuning LLMs from Basics to Breakthroughs: An Exhaustive Review of Technologies, Research, Best Practices, Applied Research Challenges and Opportunities. *arXiv preprint arXiv:2408.13296*, 2024. https://arxiv.org/pdf/2408.13296.

[46] Justus A Ilemobayo, Olamide Durodola, Oreoluwa Alade, Opeyemi J Awotunde, Temitope Olanrewaju Adewumi, Olumide Falana, Adedolapo Ogungbire, Abraham Osinuga, Dabira Ogunbiyi, Ark Ifeanyi, Ikenna E Odezuligbo, and Oluwagbotemi E Edu. Hyperparameter Tuning in Machine Learning: A Comprehensive Review. *Journal of Engineering Research and Reports*, 26(6):388–395, 2024. https://journaljerr.com/index.php/JERR/article/view/1188/2360.

[47] Saahil Afaq and Dr. Smitha Rao. Significance Of Epochs On Training A Neural Network. *International Journal of Scientific & Technology Research*, 9(6):485–488, 2020. http://www.ijstr.org.

[48] Juan R. Terven, Diana M. Cordova-Esparza, Alfonso Ramirez-Pedraza, Edgar A. Chavez-Urbiola, and Julio A. Romero-Gonzalez. Loss Functions and Metrics in Deep Learning. *arXiv preprint arXiv:2307.02694*, October 2024. https://arxiv.org/abs/2307.02694v4.

[49] Hao Li, Gopi Krishnan Rajbahadur, Dayi Lin, Cor-Paul Bezemer, and Zhen Ming (Jack) Jiang. Keeping Deep Learning Models in Check: A History-Based Approach to Mitigate Overfitting. *arXiv preprint arXiv:2401.10359*, 2024. https://arxiv.org/pdf/2401.10359.

[50] M. D. Lieberman, H. Samet, and J. Sankaranarayanan. Geotagging with local lexicons to build indexes for textually-specified spatial data. In *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*, pages 201–212. IEEE, 2010.

[51] Nikhil Ketkar. Stochastic Gradient Descent. In *Deep Learning with Python*, pages 113–132. Apress, Berkeley, CA, 2017.

[52] Nitish Shirish Keskar, Jorge Nocedal, Ping Tak Peter Tang, Dheevatsa Mudigere, and Mikhail Smelyanskiy. On large-batch training for deep learning: Generalization gap and sharp minima. In *International Conference on Learning Representations (ICLR)*, 2017. https://arxiv.org/pdf/1609.04836.

Anhang A

Diagramme

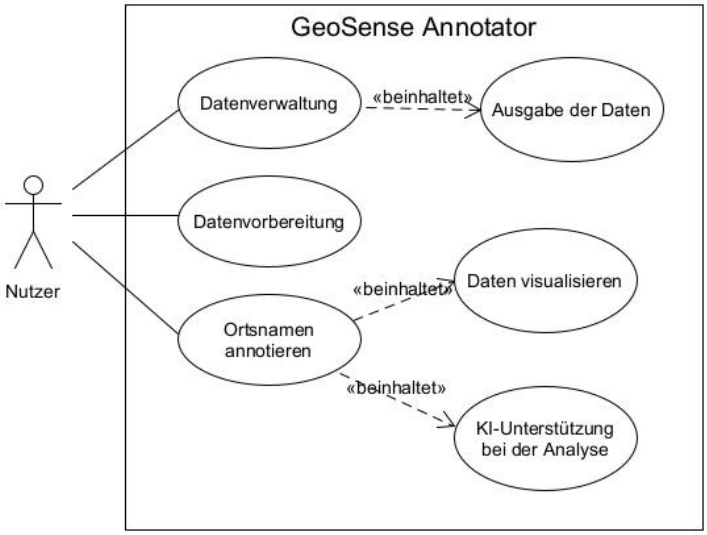


Abbildung A.1. Use-Case-Diagramm der Hauptseite

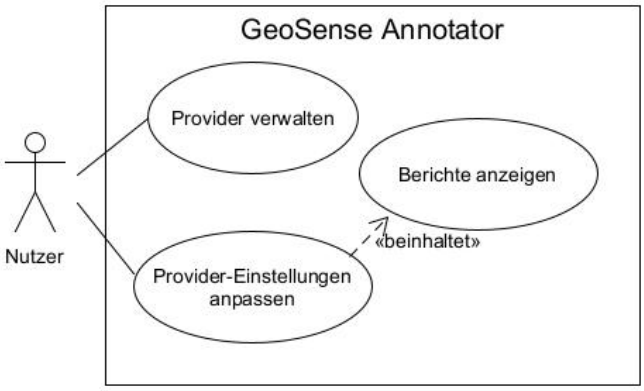


Abbildung A.2. Use-Case-Diagramm der Provider-Seite

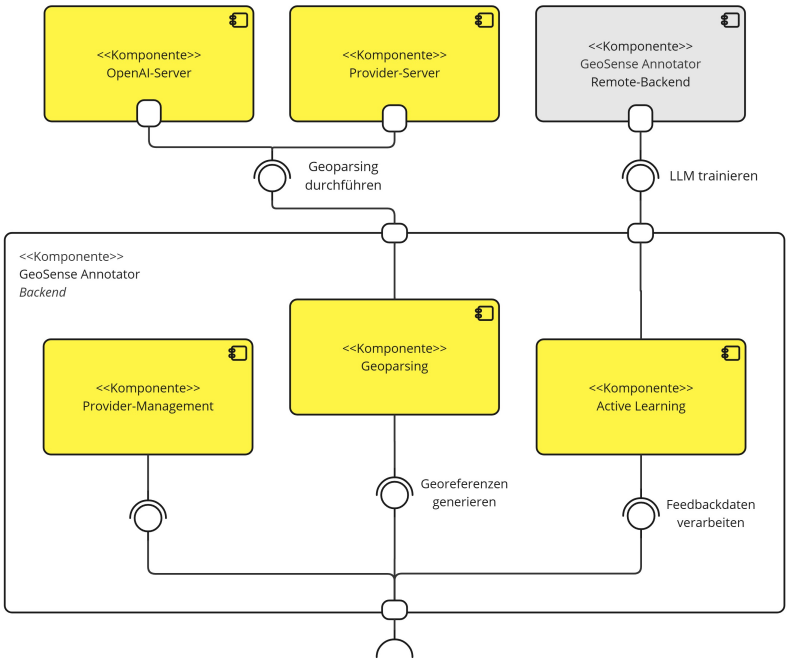


Abbildung A.3. Komponentendiagramm vom Backend des GeoSense Annotators

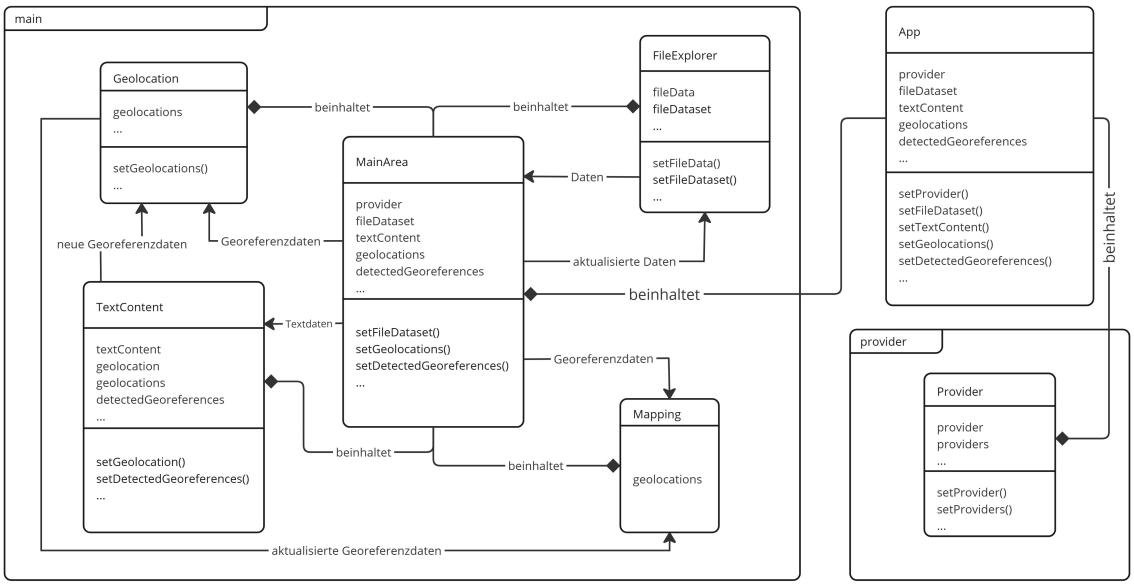


Abbildung A.4. Klassendiagramm vom Frontend des GeoSense Annotators

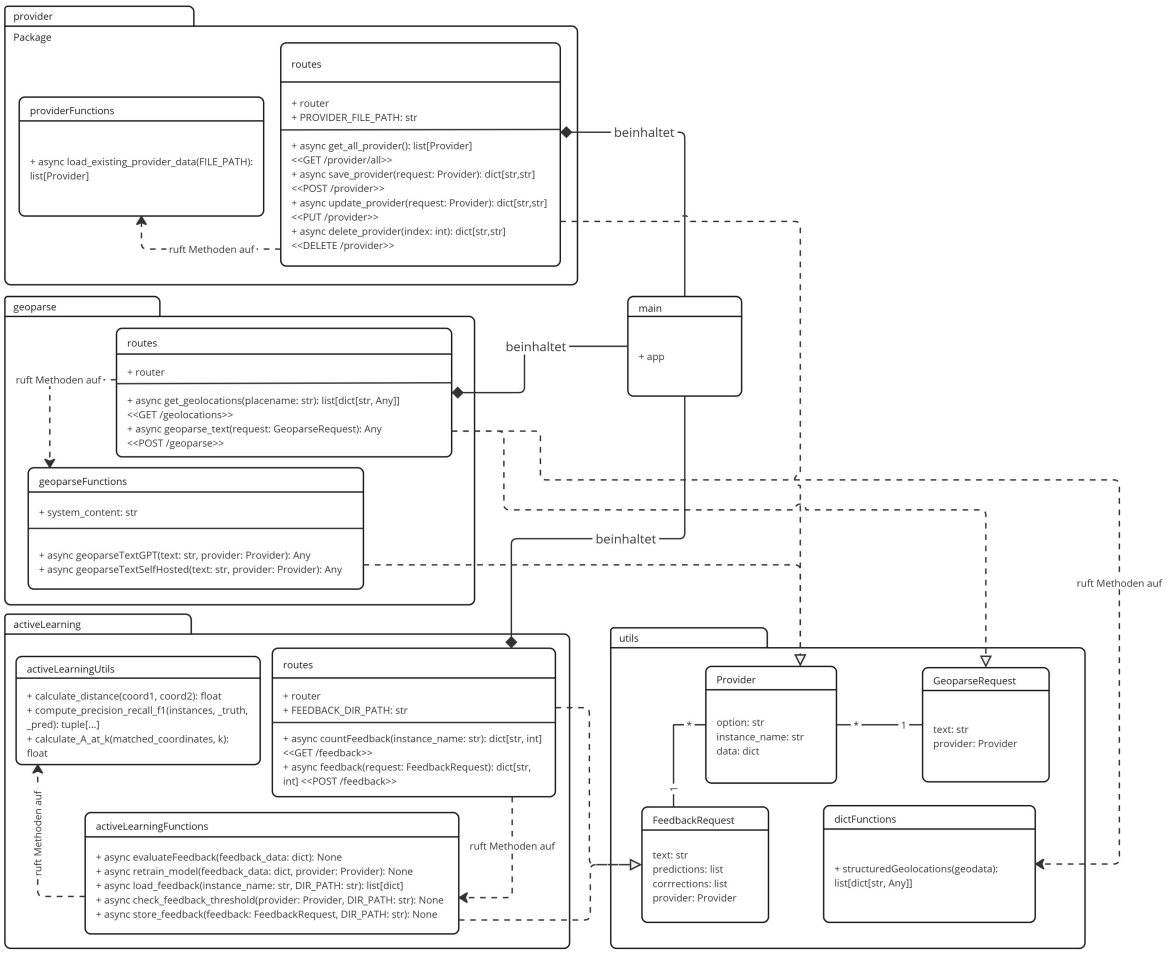


Abbildung A.5. Klassendiagramm vom Backend des GeoSense Annotators

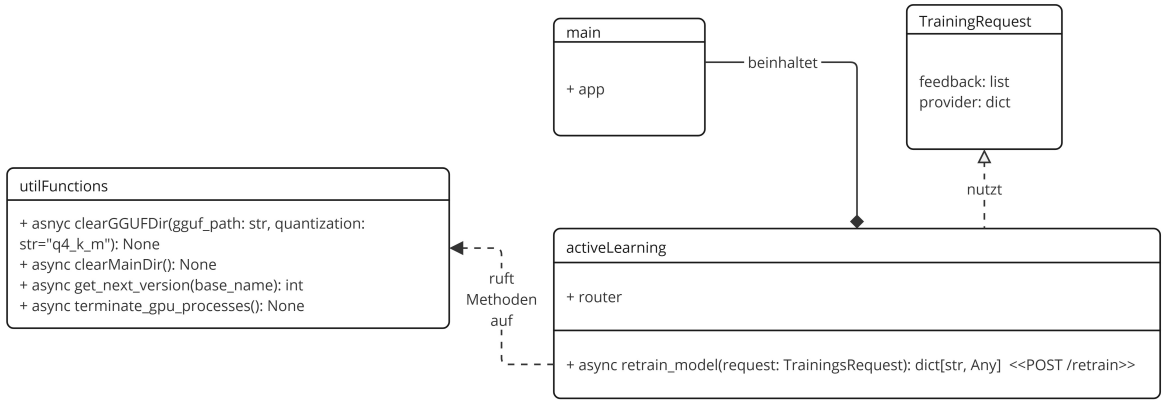


Abbildung A.6. Klassendiagramm vom Remote-Backend des GeoSense Annotators

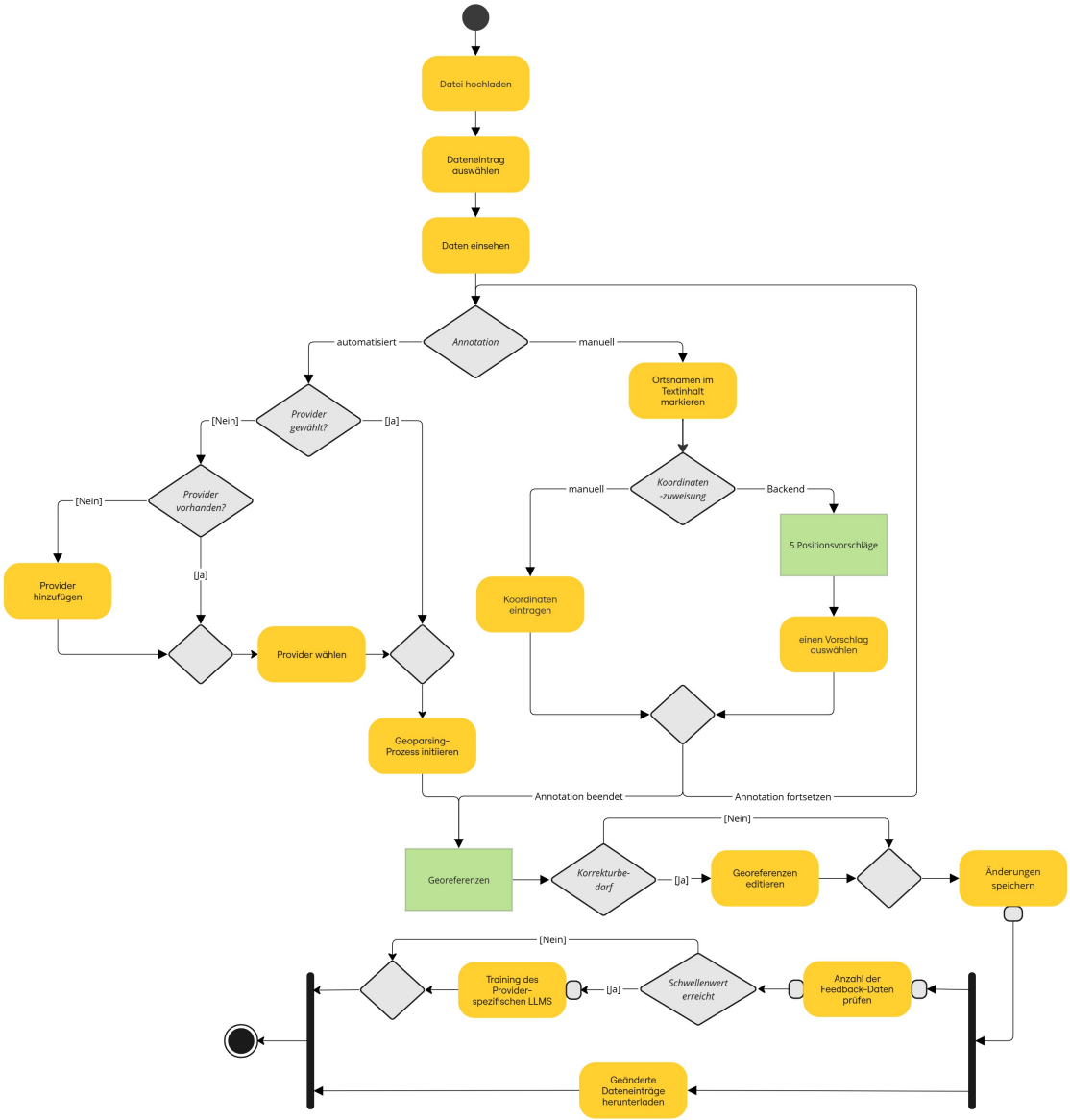


Abbildung A.7. Beispielhaftes Aktivitätsdiagramm des GeoSense Annotators

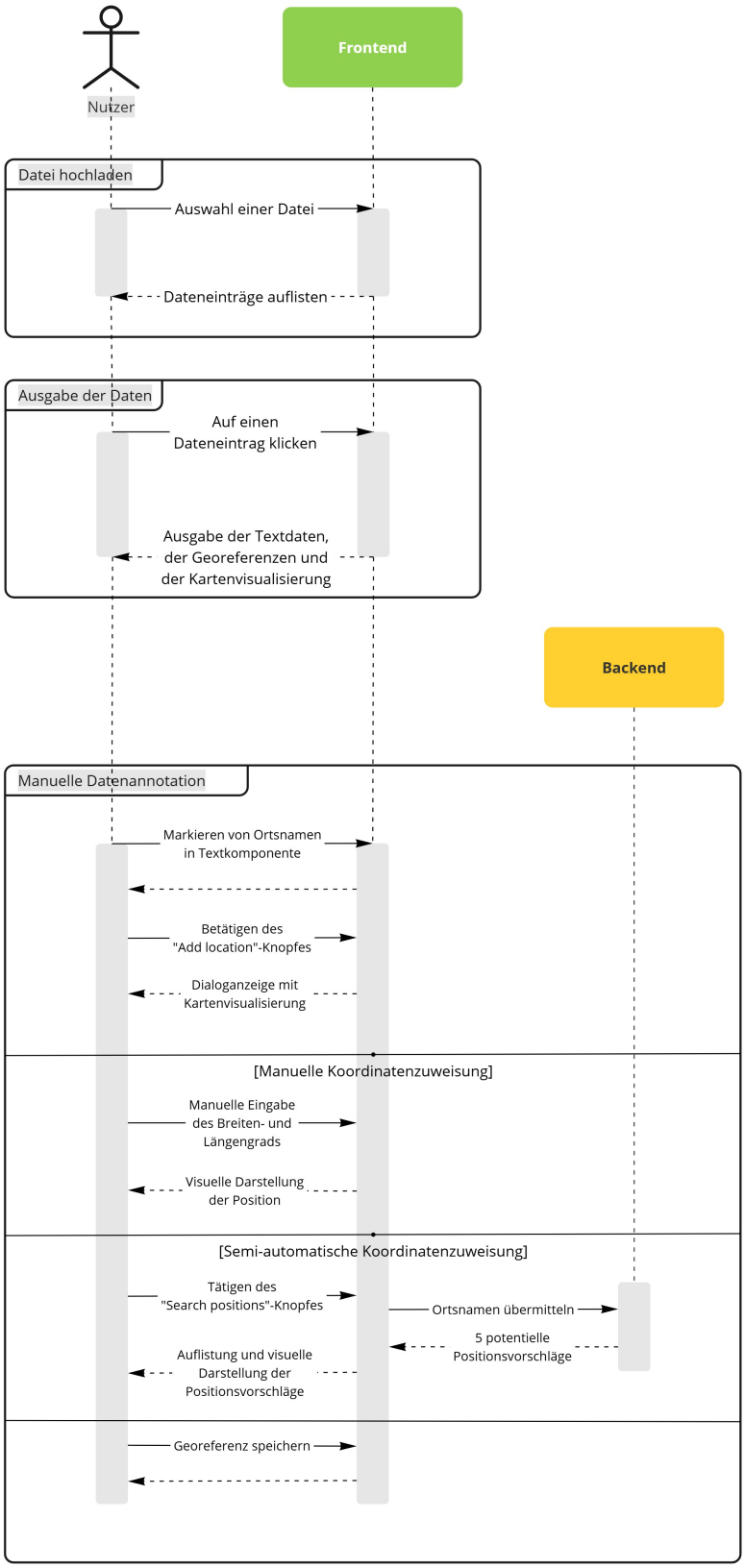


Abbildung A.8. Beipielhaftes Sequenzdiagramm des GeoSense Annotators

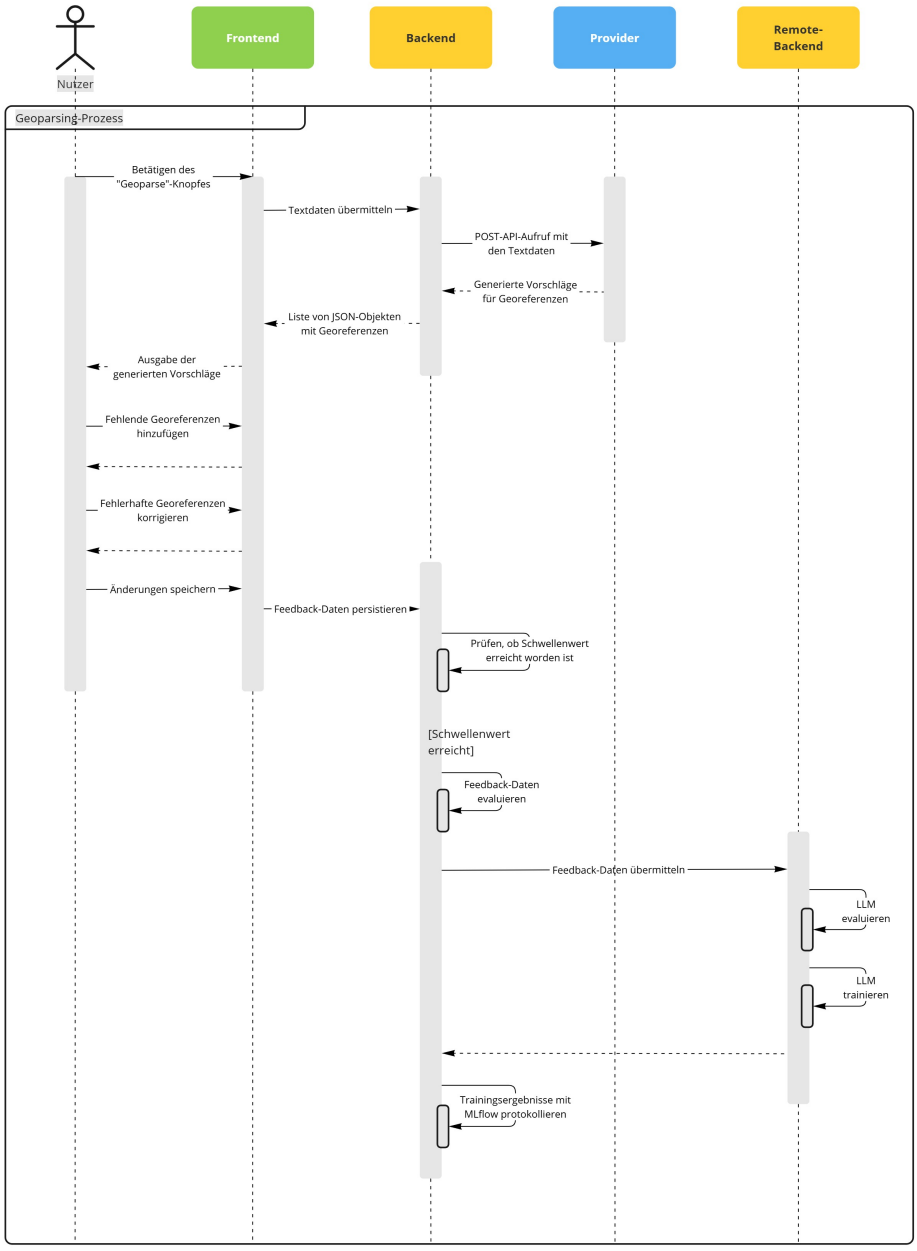


Abbildung A.9. Der Geoparsing-Prozess dargestellt als Sequenzdiagramm des GeoSense Annotators

Anhang B

Codeausschnitte

```python
async def geoparseTextSelfHosted(text: str, provider: dict):
    response = requests.post(
        url=provider["data"]["hostserver_url"] + '/chat/completions',
        json={
            "model": provider["data"]["model"],
            "messages": [
                {
                    "role": "system",
                    "content": system_content,
                },
                {
                    "role": "user",
                    "content": text,
                }
            ],
            "response_format": {
                "type": "json_schema",
                "json_schema": {
                    "name": "georeferences",
                    "strict": "true",
                    "schema": {
                        "type": "array",
                        "items": {
                            "type": "object",
                            "properties": {
                                "name": {
                                    "type": "string"
                                },
                                "position": {
                                    "type": "array",
                                    "items": {
                                        "type": "number",
                                    },
                                    "minItems": 2,
                                    "maxItems": 2
                                }
                            },
                            "required": ["name", "position"]
                        }
                    }
                }
            },
            "temperature": provider["temperature"],
            "max_tokens": -1,
            "stream": False
        },
        headers={"Content-Type": "application/json"},
    )
    output=response.json()

    return json.loads(output['choices'][0]['message']['content'])
```

Abbildung B.1. POST-Anfrage an den Provider-Hostserver zum Geoparsen von Webdaten

```
trainer = SFTTrainer(
    model = model,
    tokenizer = tokenizer,
    dataset_text_field = "text",
    train_dataset = dataset,
    eval_dataset = dataset,
    max_seq_length = 2048,
    dataset_num_proc = 2,
    packing = False,
    args = TrainingArguments(
        per_device_eval_batch_size = 2,
        do_eval=False,
        per_device_train_batch_size = 2,
        gradient_accumulation_steps = 4,
        warmup_steps = 5,
        num_train_epochs = 2,
        learning_rate = 2e-4,
        fp16 = not is_bfloat16_supported(),
        bf16 = is_bfloat16_supported(),
        logging_steps = 1,
        optim = "adamw_8bit",
        weight_decay = 0.01,
        lr_scheduler_type = "linear",
        seed = 3407,
        output_dir = "outputs"
    )
)

trainer_eval_stats = trainer.evaluate() if "finetuned" in model_name else None

trainer_train_stats = trainer.train()
```

Abbildung B.2. Training eines Modells und Bewertung der Modellleistung mithilfe des SFTTrainers

```python
async def retrain_model(feedback_data, provider) -> None:
    'Trigger retrain-job of model'

    response = requests.post(
        url='http://127.0.0.1:8571/api/retrain',
        json={
            "feedback": feedback_data,
            "provider": provider
        },
        headers={"Content-Type": "application/json"},
    )
    output=response.json()

    trainer_train_stats = output["trainer_train_stats"]
    trainer_eval_stats  = output["trainer_eval_stats"]
    trainer_args        = output["trainer_args"]

    # MLFlow Tracking
    with mlflow.start_run(
        run_name="Retrain-Job",
        tags={"job": "retrain"},
        description="Retrain-Job of configured model in provider"
    ):
        mlflow.log_param("learning_rate", trainer_args.learning_rate)
        mlflow.log_param("batch_size", trainer_args.per_device_train_batch_size)
        mlflow.log_param("epoch", trainer_args.num_train_epochs)

        mlflow.log_metric("train_loss", trainer_train_stats.train_loss)
        mlflow.log_metric("eval_loss", trainer_eval_stats.eval_loss)
```

Abbildung B.3. POST-Anfrage für den Trainingsprozess und Protokollieren der Ergebnisse mit MLflow

Anhang C

Sonstiges

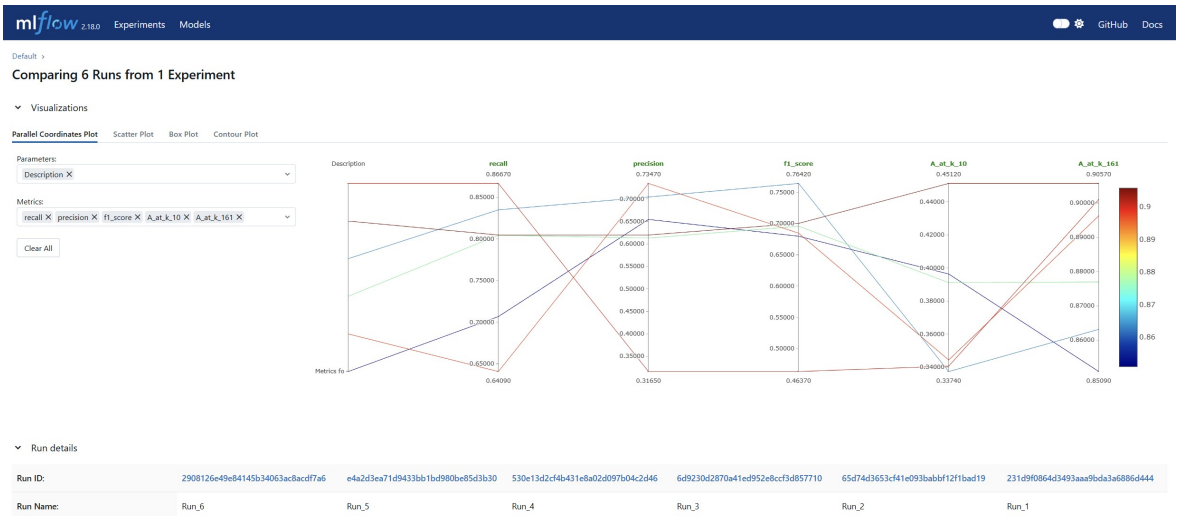


Abbildung C.1. Visuelle Darstellung der protokollierten Metriken mit MLflow