Fakultät Informatik

Automatische Generierung von themenspezifischen Podcasts aus einer großen Audiodatenbank

Bachelorarbeit im Studiengang Medieninformatik

vorgelegt von

Vincent Neumann

Matrikelnummer 358 5287

Erstgutachter:	Prof. Dr. Florian Gallwitz
Zweitgutachter:	Prof. Dr. Anna Kruspe
Betreuer:	Florian Thoma
Unternehmen:	Public Value Technologies GmbH

Hinweis: Diese Erklärung ist in alle Exemplare der Abschlussarbeit fest einzubinden. (Keine Spiralbindung)

TECHNISCHE HOCHSCHULE NÜRNBERG
GEORG SIMON OHM

Prüfungsrechtliche Erklärung der/des Studierenden

Angaben des bzw. der Studierenden:

Name: Vorname: Matrikel-Nr.:

Fakultät: Studiengang:

Semester:

Titel der Abschlussarbeit:

Ich versichere, dass ich die Arbeit selbständig verfasst, nicht anderweitig für Prüfungszwecke vorgelegt, alle benutzten Quellen und Hilfsmittel angegeben sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Ort, Datum, Unterschrift Studierende/Studierender

Erklärung der/des Studierenden zur Veröffentlichung der vorstehend bezeichneten Abschlussarbeit

Die Entscheidung über die vollständige oder auszugsweise Veröffentlichung der Abschlussarbeit liegt grundsätzlich erst einmal allein in der Zuständigkeit der/des studentischen Verfasserin/Verfassers. Nach dem Urheberrechtsgesetz (UrhG) erwirbt die Verfasserin/der Verfasser einer Abschlussarbeit mit Anfertigung ihrer/seiner Arbeit das alleinige Urheberrecht und grundsätzlich auch die hieraus resultierenden Nutzungsrechte wie z.B. Erstveröffentlichung (§ 12 UrhG), Verbreitung (§ 17 UrhG), Vervielfältigung (§ 16 UrhG), Online-Nutzung usw., also alle Rechte, die die nicht-kommerzielle oder kommerzielle Verwertung betreffen.

Die Hochschule und deren Beschäftigte werden Abschlussarbeiten oder Teile davon nicht ohne Zustimmung der/des studentischen Verfasserin/Verfassers veröffentlichen, insbesondere nicht öffentlich zugänglich in die Bibliothek der Hochschule einstellen.

Hiermit ☐ genehmige ich, wenn und soweit keine entgegenstehenden Vereinbarungen mit Dritten getroffen worden sind,

☐ genehmige ich nicht,

dass die oben genannte Abschlussarbeit durch die Technische Hochschule Nürnberg Georg Simon Ohm, ggf. nach Ablauf einer mittels eines auf der Abschlussarbeit aufgebrachten Sperrvermerks kenntlich gemachten Sperrfrist

von Jahren (0 - 5 Jahren ab Datum der Abgabe der Arbeit),

der Öffentlichkeit zugänglich gemacht wird. Im Falle der Genehmigung erfolgt diese unwiderruflich; hierzu wird der Abschlussarbeit ein Exemplar im digitalisierten PDF-Format auf einem Datenträger beigefügt. Bestimmungen der jeweils geltenden Studien- und Prüfungsordnung über Art und Umfang der im Rahmen der Arbeit abzugebenden Exemplare und Materialien werden hierdurch nicht berührt.

Ort, Datum, Unterschrift Studierende/Studierender

Datenschutz: Die Antragstellung ist regelmäßig mit der Speicherung und Verarbeitung der von Ihnen mitgeteilten Daten durch die Technische Hochschule Nürnberg Georg Simon Ohm verbunden. Weitere Informationen zum Umgang der Technischen Hochschule Nürnberg mit Ihren personenbezogenen Daten sind unter nachfolgendem Link abrufbar: **https://www.th-nuernberg.de/datenschutz/**

DokID: SB_0050_FO_Erklärung_zur_Veröffentlichung_der_Abschlussarbeit_prüfungsrechtliche Erklärung_public Vers. 2, 22.09.2017, A. Härtel (4.SB)

Kurzdarstellung

In dieser Arbeit wird ein Ansatz für die automatische Erstellung einer Podcast-Episode aus vorhandenem Audiomaterial vorgestellt. Dafür wird ein umfangreiches System aus mehreren Microservices erstellt, das den kompletten Verlauf von der Beschaffung der Audiodaten bis zur Bereitstellung einer Podcast-Episode in einem User-Interface abdeckt. Der Fokus liegt auf dem Einsatz von Large Language Models (LLMs), die in verschiedenen Stadien des Entwicklungsprozesses genutzt werden. Die Arbeit umfasst die gesamte Prozesskette, von der Datenbeschaffung und -verarbeitung bis hin zur Audioproduktion und Auslieferung des fertigen Podcasts. Außerdem wird eine Evaluation verschiedener Embedding-Methoden durchgeführt, um deren Effektivität und Eignung für das System zu analysieren.

Inhaltsverzeichnis

1 Einleitung **1**
1.1 Motivation 1
1.2 Zielsetzung 2
1.3 Überblick 2

2 Verwandte Literatur **4**

3 Ausgangslage und theoretische Grundlagen **6**
3.1 Information Retrieval 6
3.2 Embeddings 6
3.2.1 Embeddings als Computersprache 6
3.2.2 Entwicklung der Embeddingverfahren 7
3.2.3 Tokenisierer 9
3.2.4 Embeddingvektoren 9
3.2.5 Ähnlichkeitsvergleiche 11
3.3 Transformer-Architektur 13
3.3.1 Sequenz-zu-Sequenz-Modelle 13
3.3.2 RNNs, LSTMs und GRUs 13
3.3.3 Transformer-Encoder 14
3.3.4 Multihead Self-Attention 14
3.4 Large Language Model 16
3.5 Chatbots 16
3.6 Hugging Face 17
3.7 Massive Text Embedding Benchmark 18

4 Datenbeschaffung und Datenspeicherung **21**
4.1 Ausgangslage 21
4.2 Beschaffung der Audiodaten 21
4.2.1 Die ARD-Audiothek 21
4.2.2 Podcastreihe radioWissen 22
4.2.3 Datenbeschaffung über die ARD-Audiothek-API 23
4.2.4 Audiodatenanalyse 23
4.3 Transkription der Podcasts (ASR) 24

4.4 Vorgehensweise . . . 24
4.4.1 Transkriptionsmethoden im Überblick . . . 25
4.4.2 Fraunhofer IAIS . . . 25
4.4.3 Whisper . . . 25
4.4.4 Eurovox . . . 26
4.5 Datenaufbereitung . . . 27
4.5.1 Segmentbildung . . . 27
4.5.2 Satzbildung mit Whisper . . . 28
4.5.3 Satzbildung mit spaCy . . . 28
4.5.4 Segmente anpassen . . . 29
4.5.5 Segmente für TF-IDF vorbereiten . . . 29
4.6 Datenspeicherung . . . 30
4.6.1 Datenspeicherung SQLite . . . 30
4.6.2 Datenspeicherung Vektoren SQLite . . . 32
4.6.3 Datenspeicherung Dense Vektoren . . . 32
4.6.4 Datenspeicherung Sparse Vektoren . . . 33

5 Architektur . . . 35
5.1 Mikroservicearchitektur . . . 35
5.2 Anforderungen . . . 35
5.3 Programmiersprache Python . . . 36
5.4 Transkript-Segment Ranking . . . 37
5.4.1 Ranking mit Embeddings . . . 37
5.4.2 Anreicherung der Segmente . . . 37
5.4.3 Reranking mit ChatGPT . . . 38
5.5 Audio-Zusammensetzung . . . 39
5.6 Weiterführende Themenvorschläge . . . 39
5.7 Auslieferung . . . 40
5.7.1 API . . . 40
5.7.2 Design der Webseite . . . 41

6 Evaluation verschiedener semantischer Verfahren . . . 42
6.1 Ausgangslage . . . 42
6.2 Bewertungsmethoden . . . 42
6.3 Auswahl der Embedding-Modelle . . . 43
6.4 Vorgehensweise . . . 43
6.5 Auswertung . . . 44
6.5.1 Auswertung topical . . . 44
6.5.2 Auswertung re-finding . . . 45
6.6 Diskussion . . . 46

6.7 Schlussfolgerung 47
6.7.1 Kritische Reflexion 48

7 Ausblick **50**

8 Zusammenfassung **51**

Abbildungsverzeichnis **52**

Anhang **53**

Literatur **59**

Glossar **65**

Kapitel 1

Einleitung

1.1 Motivation

Podcasts sind für viele Menschen ein wichtiges Medium, um sich die Zeit zu vertreiben und sich über verschiedene Themen zu informieren. In Deutschland hören etwa 29% der Menschen regelmäßig Podcasts [60]. In dieser Arbeit liegt der Fokus auf den Personen, die Podcasts hören, um sich zu verschiedenen Themen weiterzubilden. Mittlerweile gibt es zu fast jedem Thema einen bestimmten Podcast, allein in Deutschland über 90.000 [50]. Diese Menge an Podcasts ist für viele Menschen schwer zu überblicken. Viele Suchfunktionen basieren nur auf den Metadaten der verschiedenen Podcasts und so werden im Zweifel nur Episoden gefunden, deren Titel oder Schlagworte am besten zur Anfrage passen [52]. Oft passen diese Informationen aber leider nicht zu den Anfragen der Nutzenden. In diesem Fall kann es besser sein, die tatsächlichen Inhalte einer Episode zu durchsuchen und den Nutzenden nur die für sie relevanten Ausschnitte zu liefern. Außerdem gibt es kaum Möglichkeiten, die Informationen aus verschiedenen Podcasts zu bündeln und den Hörern eine Zusammenstellung verschiedener Audiosegmente aus verschiedenen Podcasts anzubieten.

In dieser Bachelorarbeit wird untersucht, wie aus umfangreichem Audiomaterial aus Radioprogrammen oder Podcasts ein eigener Podcast zusammengestellt werden kann, der relevante Ausschnitte aus einer Vielzahl von Quellen enthält.

Ein möglicher Anwendungsfall wäre eine Person, die sich über das Thema „Überfischung der Meere“ informieren möchte und dafür genau 20 Minuten während einer Autofahrt einplant. Das System erstellt nun einen Zusammenschnitt aus verschiedenen Podcast-Episoden zu diesem Thema, der 20 Minuten lang ist, und stellt ihn der Person zur Verfügung. Der Vorteil für die Nutzenden liegt darin, dass sie selbst das Thema auswählen und die exakte Länge festlegen können. Außerdem wird das Thema von verschiedenen Sprechern aus unterschiedlichen Perspektiven erklärt, was zu einer Steigerung der Aufmerksamkeit führt [42].

1.2 Zielsetzung

Diese Bachelorarbeit ist in zwei Teile aufgeteilt. Im ersten Teil der Arbeit wird beschrieben, wie ein Prototyp für ein System zur automatischen Podcastgenerierung aufgebaut sein könnte. Dazu werden die einzelnen Mikroservices vorgestellt und die Wahl der verwendeten Technologien begründet. In einigen Fällen werden mehrere unterschiedliche Services miteinander verglichen. In anderen Fällen wird beschrieben, wie bei der Entstehung des Projektes bestimmte Technologien durch andere ersetzt wurden, da in der Zwischenzeit neue Erkenntnisse gewonnen worden sind. Für die Auswahl der passenden Audiosegmente werden verschiedene Methoden des Natural Language Processing verwendet, insbesondere Text-Embeddings. Außerdem werden die Fähigkeiten von Large Language Models genutzt, um die Qualität der generierten Podcast-Episoden zu verbessern. Für die Interaktion mit dem Benutzer soll außerdem eine grafische Benutzeroberfläche bereitgestellt werden, die den Nutzenden die Auswahl eines Themas und die Länge der Podcast-Episode ermöglicht.

Im zweiten Teil der Arbeit geht es insbesondere um die Auswahl der richtigen Embeddings. Dabei wird die Fähigkeit, relevante Segmente zu finden, anhand verschiedener Metriken evaluiert. Es werden die Ausgaben verschiedener Embeddings miteinander verglichen und die Güte mithilfe eines Large Language Models bewertet.

1.3 Überblick

Diese Arbeit ist in acht Kapitel gegliedert.

Im ersten Kapitel werden die Zielsetzung, Motivation und ein kurzer Überblick dargelegt, während im zweiten Kapitel die relevante Literatur zusammengefasst wird.

Das dritte Kapitel erklärt die theoretischen Grundlagen, die für das Verständnis der später aufgeführten Technologien erforderlich sind. Besonderes Augenmerk wird dabei auf die Technologien der Embeddings und der Large Language Models gelegt.

Im vierten Kapitel geht es um die Beschaffung der Audiodaten, wobei verschiedene Methoden zur Transkription dieser Audiodaten beschrieben und begründet werden, welche Transkriptionsmethode für diese Arbeit gewählt wurde. Außerdem werden effiziente Wege für die Datenspeicherung diskutiert.

Die Architektur des Systems wird im fünften Kapitel dargestellt, wobei ein besonderer Fokus auf die semantische Analyse der Transkriptionen gelegt und verschiedene Aspekte des Natural Language Processing sowie der Large Language Models erörtert werden.

Im sechsten Kapitel werden verschiedene Methoden zur semantischen Analyse evaluiert, die verwendeten Modelle, die Wahl der Anzahl und Länge der Abschnitte sowie die finale Zusammensetzung dieser erklärt und begründet.

Ein Ausblick auf zukünftige Weiterentwicklungen wird im siebten Kapitel gegeben und im letzten Kapitel werden die Ergebnisse dieser Arbeit zusammengefasst.

Kapitel 2

Verwandte Literatur

Im Bereich der automatisierten Podcasterstellung gibt es einige Versuche mittels künstlich generierter Texte und Stimmen einen eigenen Podcast zu erstellen.

In dem Artikel „NewsPod: Automatic and Interactive News Podcasts“ [45] wird ein neuer Ansatz für die automatische Erstellung von Podcast-Episoden vorgestellt. Dazu wird ein interaktiver Sprachbot entwickelt, der zu bestimmten Nachrichten Fragen beantworten kann und mithilfe einer synthetisch generierten Stimme mit den nutzenden Personen interagiert. Mehrere Machine Learning Systeme ermitteln den Inhalt von bestimmten Nachrichtenseiten und extrahieren daraus Fragestellungen, die den Inhalt des Textes widerspiegeln sollen. Die Autoren benutzen ein GPT-2 Sprachmodell, das auf Fragengenerierung trainiert wurde, um aus jedem Absatz 7 Fragen zu ermitteln. Ein weiteres Question-Answering Sprachmodell wird dann trainiert, um die Ausschnitte in den Artikeln zu finden, welche diese Fragen beantworten. Der Retrieval Step wird hier also nicht durch Embeddings, sondern durch ein Sprachmodell vollzogen.

Der wichtigste Aspekt dieses Artikels ist, dass die Anwendenden interaktiv mit der Software agieren und selbstgewählte Fragen mithilfe eines Mikrofons oder einer Tastatur stellen können. Das Question-Answering Sprachmodell versucht dafür relevante Segmente aus mehreren Nachrichtenartikeln zu finden, die die Fragen der nutzenden Person beantworten. Diese werden dann mithilfe einer synthetisch generierten Stimme von Googles Text-To-Speech API vorgelesen.

Die Autoren dieses Papers führten außerdem zwei Studien zur Nutzung dieses Systems durch. Der Gegenstand der ersten Studie ist, inwieweit die Zufriedenheit einer Testgruppe mit der Erzählweise des Textes und der automatisch generierten Stimme des Sprechers korreliert. Darin konnten die Autoren feststellen, dass einer ihrer Ansätze, QA Best, so erfolgreich war, dass 80% der Testpersonen angaben, dass sie dieses System in Zukunft nutzen würden, um Nachrichten zu konsumieren. Die zweite Studie untersucht die Interaktion der Zuhörenden während der Benutzung des Systems. Diese Studie kam zu dem Schluss, dass zwar die Bereitschaft eigene Fragen zu stellen mit 85% der Zuhörenden sehr hoch war, die

Zufriedenheit der Testenden mit der Qualität der Antworten aber sehr gering ausfiel, da 76% der Antworten als verwirrend eingestuft wurden. [45]

In einem weiteren für diese Arbeit relevanten Paper [40] werden die Ergebnisse der Text Retrieval Conference (TREC) 2020 für die Podcastanalyse vorgestellt. Für diese Aufgabe analysierten die Teilnehmer einen großen Datenbestand an Podcasttranskripten. Die Teilnehmer versuchten aus über 100.000 Podcasttranskripten die wichtigsten Segmente zu verschiedenen Themen herauszufiltern. Im Vorfeld wurden die Transkripte durch ein System zur Automatic Speech Recognition (ASR) erstellt. Jedes Transkriptsegment ist dabei jeweils genau zwei Minuten lang und überlappt sich um eine Minute mit einem folgenden Segment. Die entsprechenden Aufgabenstellungen sind in drei Kategorien eingeteilt: „topical“, „re-finding“ und „known items“. Bei der Kategorie „topical“ wird verlangt, passende Segmente zu einem bestimmten Thema zu finden. Beim „re-finding“ geht es darum, einen zuvor bekannten Audioinhalt wiederzufinden. Dabei waren nur Teile des Audioinhaltes oder bestimmte Rahmenbedingungen vorgegeben (z.B. eine Podcast-Episode, welche die Person vor einer Woche hörte). In der letzten Kategorie den „known items“ sind bereits der Titel bzw. weitere Metainformationen bekannt. Die einzelnen Themen verfügten dabei außerdem noch über eine Beschreibung, die spezifiziert, was eine Testperson von der Suche mit diesem Prompt erwarten würde.

Insgesamt nahmen neun Teilnehmer an dieser Aufgabe teil und reichten eine Lösung ein. Darunter Universitäten aus den USA und Dublin sowie ein Team des Musikstreaminganbieters Spotify. Die Evaluation der Ergebnisse wurde manuell durchgeführt. Verschiedene Gutachter des TREC bewerteten die Ergebnisse der Teilnehmer auf einer Skala von 5 (perfekt) bis 0 (nicht passend). Am besten schnitten dabei die Lösungen der Universität Maryland ab, die zur Datenaufbereitung eine Mischung aus Stemming und Word2Vec benutzten und für die Retrievalfunktion die Search Engine Indri einsetzten. Die Indri Search Engine wurde von der University of Massachusetts und Carnegie Mellon University aufgebaut, wird aber zurzeit nicht mehr weiterentwickelt [49].

Speziell für deutsche Audioinhalte ist ein Artikel zu erwähnen, indem das Fraunhofer-Institut für Intelligente Anlyse- und Informationssysteme (IAIS) 2015 die Audiominig Plattform medas vorstellt [52]. Diese soll für die ARD-Audiothek die Suchfunktionen für verschiedene Audioinhalte verbessern. Dazu erstellten die Forscher ein System, dass diese Inhalte automatisch mit ASR transkribieren, eine Spracherkennung durchführen und Keywörter extrahieren kann. Diese Daten sind über eine REST Schnittstelle abrufbar. Allerdings sind viele der beschriebenen Technologien mittlerweile nicht mehr State-of-the-Art und spielen deswegen in dieser Arbeit kaum eine Rolle. [52] In Kapitel 5 wird darauf näher eingegangen.

Kapitel 3

Ausgangslage und theoretische Grundlagen

3.1 Information Retrieval

Das Ziel dieser Arbeit besteht darin, eine Schnittstelle für einen Zuhörer zu bieten, die automatisch Podcast-Episoden erstellen kann. Die generierte Podcast-Episode besteht dabei aus einem Zusammenschnitt aus verschiedenen bereits vorhandenen Podcast-Episoden, die alle zu dem vom Zuhörer gewünschten Thema passen. Die wichtigste Aufgabe besteht also darin, die wesentlichen Abschnitte aus den Episoden herauszufinden. Dazu muss das System verstehen, welche Abschnitte zu verschiedenen Anfragen passen.

Das Thema Information Retrieval (Informationsrückgewinnung) bezeichnet den Vorgang, aus einer großen Menge unsortierter Daten bestimmte Informationen wieder zu extrahieren [11]. Im Gegensatz zum Data-Mining werden dabei keine neuen Daten erschaffen, sondern lediglich bereits existierendes Wissen wieder zur Verfügung gestellt. Ein bekanntes Beispiel eines Information Retrieval Algorithmus ist der PageRank-Algorithmus für die Suchmaschine Google [67]. Suchmaschinen sind das Paradebeispiel eines Information Retrieval Systems. Ein User gibt Stichworte ein, über die er mehr Informationen erhalten möchte und erhält daraufhin Links zu Webseiten, die diese Informationen wahrscheinlich enthalten.

In dieser Arbeit wird die Informationsrückgewinnung auf Basis der Daten von Podcasttranskripten betrieben. Dazu werden Techniken der Text Embeddings benutzt, um relevante Informationen zu einem bestimmten Thema aus einer großen Anzahl an Dokumenten zurückzugewinnen.

3.2 Embeddings

3.2.1 Embeddings als Computersprache

Text Embeddings sind ein Weg, die menschliche Sprache für Computer verständlich zu machen. Die menschliche Sprache ist ein hochkomplexes Konstrukt mit einer Grammatik, die

sehr viel flexibler, kreativer, vieldeutiger und komplexer ist als die Maschinensprache. Es gibt viele kleine Bedeutungsnuancen, die Sprache ist stark von dem allgemeinen Wissensstand der Welt geprägt und sie ändert sich im Laufe der Zeit. Das alles macht es für Computer sehr schwierig, die menschliche Sprache zu verstehen. Neuere Forschung im Bereich des Natural Language Processing bietet einige Ansätze, um dieses Problem zu lösen. Es gibt dazu viele verschiedene Methoden, die alle darauf abzielen, Worte oder Texte in Vektoren umzuwandeln, die den Inhalt dieser Worte oder dieser Texte repräsentieren [53] [70]. Diese Vektoren nennt man Embeddings. Das Ziel dieser Embeddings besteht darin, eine Suchfunktion zu beschreiben, die auf eine Anfrage (Query) hin passende Dokumente aus einem großen Korpus an Dokumenten liefern kann.

3.2.2 Entwicklung der Embeddingverfahren

Die ersten Algorithmen mit dem Ziel, eine große Anzahl von Dokumenten durchsuchen zu können, waren Suchalgorithmen wie TF-IDF [82] oder BM25 [31], die in den 1970er Jahren aufkamen. Diese versuchten, Worthäufigkeiten als Anhaltspunkt zu nehmen, um Ähnlichkeiten zwischen Dokumenten zu erkennen und Suchfunktionen aufzubauen.

Semantische Analysen begannen erst ab ca. 1990 mit der Latenten Semantischen Analyse (LSA) [46]. Diese verwendet Eigenvektoren, um aus einer Term-Frequency Matrix versteckte (latente) Eigenschaften zu ermitteln, welche die Dokumente besser repräsentieren als die TF-Matrix an sich. Es werden von diesen Eigenvektoren nur die wichtigsten ausgewählt, sodass jedes Dokument sehr gut als eine Linearkombination dieser Vektoren repräsentiert werden kann. So werden nur noch die Koeffizienten dieser Vektoren gespeichert und man kann von einer Dimensionsreduktion profitieren. Die resultierenden Koeffizienten-Vektoren sind nun nicht mehr spärlich besetzt, sondern dicht, und die resultierenden Eigenvektoren bilden häufig latente Themen der Dokumente ab.

Semantische Word Embeddings erlangten erst mit der Veröffentlichung von Word2Vec [53] im Jahr 2013 eine breitere Nutzung. Word2Vec verwendet ein neuronales Netz, um mithilfe der benachbarten Wörter Kontextinformationen über das eigentliche Wort zu erhalten. Dafür gibt es die beiden Ansätze Continuous Bag of Words und Skip-Gram. Beim Continuous Bag of Words-Ansatz erhält das neuronale Netz die umgebenden Wörter als Input und das Modell hat die Aufgabe, daraus das Zielwort zu ermitteln. Dieses Verfahren wird in einem Sliding Window für jedes Wort aus einem Text wiederholt. Für einen bestimmten Korpus aus verschiedenen Dokumenten kann dieses Modell dadurch die Beziehungen verschiedener Worte zueinander und die Ähnlichkeit verschiedener Worte, die oft im gleichen Kontext vorkommen, erlernen. Das Verfahren des Skip-Gram funktioniert ähnlich, allerdings wird dabei für jedes Wort versucht, die umliegenden Worte zu ermitteln. Dieser Ansatz dauert

länger im Training, hat aber den Vorteil, dass er auch seltener vorkommende Worte gut repräsentieren kann. [53]

Ein Jahr später wurde GloVe (Global Vectors for Word Representation) [69] als Forschungsprojekt von der Stanford Universität vorgestellt. GloVe verbindet Konzepte aus LSA und Word2Vec, indem es auf der Faktorisierung einer globalen Matrix beruht, aber die Distanz von Worten zueinander mitberücksichtigt. Dadurch schaffte GloVe eine weitere Verbesserung der Embeddings und wird heutzutage noch unter anderem in NLP-Bibliotheken wie spaCy verwendet [35].

Im Jahre 2018 wurde der Ansatz ELMo (Embeddings from Language Models) [70] für Word Embeddings vorgestellt. Dieser benutzt bidirektionale Long Short-Term Memory Modelle (LSTM) [33], um daraus Embeddings zu generieren. LSTM-Modelle basieren auf rekurrenten neuronalen Netzen (RNN) [14] und gehören zu den Sequenz-zu-Sequenz-Modellen. Sie bestehen dabei aus einem Encoder und einem Decoder. Der Encoder codiert eine Inputsequenz, zum Beispiel ein Dokument, zu einem Embeddingvektor. Der Decoder dekodiert diesen Embeddingvektor wieder in eine Sequenz, z.B. eine Zusammenfassung des Dokuments. Für ELMo ist dabei nur der Encoder-Teil wichtig. Dabei werden zwei Layer von Forward- und Backward-LSTMs eingesetzt, die für jedes Wort den Kontext vor und nach dem Wort berücksichtigen, um semantische Embeddings zu erstellen.

Ein halbes Jahr später wurden die Erstellung von Word- und Sentence-Embeddings durch BERT (Bidirectional Encoder Representations from Transformers) [17]revolutioniert. BERT ist ein auf NLP-Aufgaben spezialisierter Transformer. Das Modell wurde von Google im Jahr 2019 entwickelt und war seinerzeit das beste Modell, da es in elf verschiedenen Aufgaben im Bereich des NLP State-of-the-Art-Ergebnisse lieferte [17]. Es ist auf zwei Datenbeständen, dem BooksCorpus mit 800 Millionen Worten und der englischen Wikipedia mit 2,5 Milliarden Worten trainiert worden. Die Aufgaben im Trainingsprozess waren die Next Sentence Prediction (NSP), bei der das Modell entscheiden muss, ob zwei Sätze wahrscheinlich zusammen vorkommen, und das Masked Language Modeling, bei dem das Modell unkenntlich gemachte Wörter in Sätzen wiederherstellen soll. Dadurch hat BERT sowohl auf Wortebene als auch auf Satzebene wichtige Merkmale der Sprache gelernt. Durch die große Auswahl an verschiedenen Themen im Trainingsprozess kann das Modell für viele verschiedene Sachverhalte sinnvolle semantische Embeddingvektoren erstellen. [17]

Seit der Entdeckung der Transformerarchitektur und der Entstehung von BERT entwickelt sich der Bereich der Embedding-Modelle rasant. Täglich werden neue Modelle publiziert und in Abständen von wenigen Wochen erreichen neue Modelle State-of-the-Art-Performance auf populären Benchmarks wie die Massive Text Embedding Benchmark [58].

3.2.3 Tokenisierer

Um Textsequenzen mithilfe von Embeddings zu verarbeiten, werden diese Texte zunächst in Tokens zerlegt, bevor sie von einem Embedding-Modell embedded werden. Ein Token kann dabei je nach Modell ein einziges Wort, mehrere Worte oder nur ein Bruchteil eines Wortes sein. [32] In Machine Learning-Anwendungen nutzt man dafür verschiedene Tokenisierer [89]. Es gibt einfache Tokenisierer, wie Whitespace-Tokenisierer, die einen Text einfach an Leerzeichen teilen und die einzelnen Wörter als Tokens behandeln. Etwas besser sind Wort-Tokenisierer, die auch Satzzeichen erkennen und dadurch Punkte und Kommata nicht an das Ende des vorangegangenen Wortes anhängen. Einen solchen Tokenisierer nutzt beispielsweise spaCy, um „Linguistic Features“ zu bestimmen [35]. Es gibt Subword-Tokenisierer, die Worte noch einmal in kleinere Abschnitte teilen. Das Byte-Pair Encoding versucht aufgrund von statistischen Methoden, häufig vorkommende zusammenhängende Zeichenfolgen als Tokens zu identifizieren. Diese Art von Tokenisierer wird am häufigsten für die Eingabe bei Transformern genutzt [89]. Die Transformermodelle von OpenAI benutzen beispielsweise tiktoken als Tokenisierer [84], das bekannte BERT-Modell von Google benutzt den WordPiece-Tokenisierer [17].

3.2.4 Embeddingvektoren

3.2.4.1 Sparse Vektoren

Embeddings können in zwei verschiedene Kategorien eingeteilt werden: dünne (sparse) Vektoren und dichte (dense) Vektoren.

Sparse Vektoren, wie der TF-IDF-Algorithmus [82] und der BM25-Algorithmus [31], versuchen syntaktische Eigenschaften von Dokumenten zu erfassen. Term Frequency-Inverse Document Frequency (TF-IDF) ist ein Algorithmus, der es erlaubt, Dokumente nach Keywords zu durchsuchen. Das Verfahren verwendet Worthäufigkeiten (Term Frequency), um Dokumente, in denen ein Wort häufiger vorkommt, zu bevorzugen. Außerdem verwendet es die generelle Auftrittsrate von Wörtern (Inverse Document Frequency), um Wörtern, die seltener vorkommen, mehr Gewicht bei der Suche einzuräumen.

Daraus bildet der TF-IDF-Algorithmus einen dünnbesetzten Vektor, der die Anzahl und Seltenheit jedes im Dokument vorkommenden Wortes abbildet. Dünnbesetzt heißt in diesem Zusammenhang, dass der Vektor viele Einträge mit dem Wert 0 besitzt. Das folgt aus der Bedingung, dass diese Vektoren untereinander vergleichbar sein müssen und dadurch Einträge für jedes Wort besitzen, das im gesamten Korpus an Dokumenten vorkommt. Jedes einzelne Dokument enthält dabei aber nur einen Bruchteil der Gesamtwörter und damit viele Nullwerte. [82]

Best Matching 25 (BM25) ist eine Familie von Algorithmen, die versuchen, den TF-IDF-Algorithmus zu verbessern, indem sie die Priorisierung langer Dokumente und zu oft vorkommende Worte ausgleichen. [31]

Außerdem kann man diese Algorithmen noch anpassen, sodass sie auch N-Gramme als Einträge der Vektoren miteinbeziehen. Das heißt, dass nicht nur einzelne Wörter berücksichtigt werden, sondern auch häufig zusammen vorkommende Wörter (N-Gramme) als einzelne Tokens in dem Vektor abgebildet werden. Das bewirkt zwar, dass die Ähnlichkeit der Dokumente in der Regel besser ermittelt werden kann, vergrößert aber die Vektoren. [18]

3.2.4.2 Dichte Vektoren

Ein relativ moderner Ansatz besteht in der Suche mithilfe von dichten Embeddingvektoren. Dabei wird versucht, die Semantik (also die Bedeutung) der einzelnen Segmente mathematisch mithilfe eines Vektors zu repräsentieren. Dieser Vektor soll dann zur Suche nach Ähnlichkeit eingesetzt werden. Ein Vektor, der aus dem Satz „Ich sitze auf einer Bank im Grünen“ besteht, und ein Vektor mit dem Satz „Im Park steht eine alte Bank“ sollen dabei sehr ähnlich zueinander sein. Der Embeddingvektor des Satzes „Ich gehe in die Bank und hebe Geld ab“ sollte dazu wenig Ähnlichkeit aufweisen. Hierbei soll erkannt werden, dass sich die ersten beiden Sätze auf eine Bank zum Sitzen in der Natur beziehen und der dritte Satz ein Geldinstitut betrifft. Ein solch filigranes Verständnis der Bedeutung ist nicht einfach zu erreichen. Ein stochastischer Algorithmus würde die Unterschiede bei solchen Homonymen in den meisten Fällen nicht erkennen können. Stochastische Algorithmen wie z.B. TF-IDF haben auch Probleme mit Negierungen, wie zum Beispiel in „Ich habe nichts gegessen.“ und „Nichts habe ich heute gemacht, außer gegessen“.

Für unsere Aufgabe ist es sinnvoll, nicht nur zu überprüfen, ob zwei Sätze ungefähr die gleiche Bedeutung haben, wie „Die Sonne bringt mich zum Schwitzen“ oder „Ich schmelze in der Hitze“. Besser wäre eine asymmetrische Ähnlichkeit, wie sie bei Frage- und Antwortpaaren vorkommt. Zum Beispiel wäre auf die Frage: „Welche Farbe hat der Himmel bei Sonnenuntergang?“ eine gute Antwort: „Der Himmel hat bei Sonnenuntergang oft orange und rosa Farbtöne.“ und nicht der Satz „Wie sieht der Himmel am Abend aus?“ Diese asymmetrische Ähnlichkeit ist schwerer zu ermitteln, da man normalerweise in der Antwort Informationen findet, die in der Frage nicht vorkommen.

3.2.4.3 Dimensionalität von Embeddingvektoren

Die Dimensionalität von verschiedenen Embeddings hat Einfluss auf deren Performance. Kleinere Embeddings verbrauchen weniger Speicherplatz und lassen sich schneller untereinander vergleichen, was den Retrieval-Prozess beschleunigt. Andererseits bietet eine kleinere

Dimensionalität auch weniger Möglichkeiten für die LLMs, semantische Informationen darin zu speichern. Das heißt aber nicht, dass größere Embeddingvektoren auch gleichzeitig die Semantik besser repräsentieren müssen. Sparse Vektoren sind meist größer als Dense Vektoren, können aber nur lexikalische Eigenschaften repräsentieren. Aber auch bei Dense Vektoren bedeuted eine Dimensionalität nicht gleich eine bessere Performance. Dies hängt vor allem damit zusammen, wie leistungsstark die Architektur des Modells ist und welche sowie wie viele Trainingsdaten zum Trainieren verwendet wurden. Das Modell voyage-lite-02-instruct belegt zum Beispiel auf dem Massive Text Embedding Benchmark Leaderboard mit einer Dimensionalität von 1024 den zweiten Platz [56]. Auf den Plätzen drei bis fünf liegen jedoch Embedding-Modelle mit einer Dimensionalität von 4096.

3.2.5 Ähnlichkeitsvergleiche

Mithilfe der Embedding-Vektoren können Sätze gefunden werden, die eine semantische Ähnlichkeit zueinander aufweisen. Zum Beispiel besitzen die Vektoren des all-MiniLM-L6-v2-Modells [54] 384 Dimensionen, haben also 384 Gleitkommazahlen gespeichert, die zwischen -1 und 1 liegen. Diese Gleitkommazahlen-Vektoren könnte man auch als Feature-Vektoren begreifen. Zum Beispiel könnte die erste Zahl dieses Vektors für die Erwähnung von Professoren in dem Satz stehen (-1 für keine Professoren; 1 für viele Professoren). Die zweite Zahl könnte für das Thema Essen stehen (-1 für wenig mit Essen zu tun; 1 für sehr viel mit Essen zu tun). Damit hätte der Satz „In der Mensa gibt es jeden Tag Currywurst mit Pommes“ an der ersten Stelle vielleicht eine 0,1, weil der Begriff „Mensa“ leicht mit Uni und Professoren konnotiert wird. An der zweite Stelle würde der Wert zum Beispiel bei 0,94 liegen, da es in dem Satz offensichtlich um das Essen handelt. In der Realität wird das Modell sehr wahrscheinlich komplexere Merkmale lernen, die für Menschen nicht so offensichtliche sind [9]. Ein Grund dafür ist, dass das Modell vor allem pro Eintrag eine Linearkombination von verschiedenen Merkmalen lernt, also jede Zahl eine Überlagerung vieler verschiedener Eigenschaften darstellt [9].

Die Vergleiche der Embeddings können mit verschiedenen Algorithmen berechnet werden. Im Bereich des Machine Learnings werden vor allem die euklidische Distanz und die Kosinusdistanz angewendet.

3.2.5.1 Euklidische Distanz

Die euklidische Distanz zwischen zwei Vektoren kann über den Satz des Pythagoras berechnet werden. Dazu wird für jede einzelne Dimension die Differenz der beiden Vektoren in dieser Dimension gebildet. Diese Differenzen werden quadriert und anschließend wird die Summe dieser Quadrate gebildet. Zum Schluss wird die Quadratwurzel aus dieser Summe

gezogen. Für die beiden Vektoren $\begin{pmatrix}0\\1\end{pmatrix}$ und $\begin{pmatrix}-1\\0\end{pmatrix}$ wäre die Euklidische Distanz $\frac{1}{\sqrt{2}}$,

da $\sqrt{(0-(-1))^2+(1-0)^2} = \frac{1}{\sqrt{2}}$

Dieses Distanzmaß ist sensibel gegenüber der Länge der Vektoren. Wenn die Vektoren nicht normiert wurden, kann die euklidische Distanz sehr groß werden und die Ergebnisse einer Ähnlichkeitssuche verfälschen. Da die Berechnung der Wurzel für den Vergleich von verschiedenen Distanzen nicht relevant ist, wird in Machine-Learning-Anwendungen häufig die quadratische L2-Norm verwendet. Diese bildet auch das Produkt der Quadrate der einzelnen Differenzen, verzichtet aber auf die Wurzel. [43]

$euclidean\_L2\_norm(a,b) = \sum_{i=1}^{n}(a_i - b_i)^2$

3.2.5.2 Kosinusdistanz

Ein weiteres Ähnlichkeitsmaß kann über den Winkel zwischen zwei Vektoren definiert werden. Dieser Winkel ist unabhängig davon, wie lang die Vektoren sind, immer gleich. Die weitverbreitetste Methode, um diesen Winkel zu messen, ist über die Kosinusähnlichkeit. Diese berechnet den Kosinus des Winkels und kann einfach über die Formel

$\text{cosine\_similarity}(\mathbf{a},\mathbf{b}) = \cos\theta = \frac{\mathbf{a}\cdot\mathbf{b}}{\|\mathbf{a}\|\|\mathbf{b}\|}$

berechnet werden. Dabei wird für beide Vektoren zunächst das Skalarprodukt bestimmt. Das Skalarprodukt besitzt die Eigenschaft, groß zu sein, wenn beide Vektoren in ähnliche Richtungen zeigen, und kleiner, wenn beide Vektoren in unterschiedliche Richtungen zeigen. Dabei hat die Länge der unterschiedlichen Vektoren einen Einfluss auf die Größe des Skalarproduktes. Um dieser Verzerrung entgegenzuwirken, wird das Skalarprodukt durch das Produkt der Beträge der Vektoren geteilt. Dadurch erhält man den Kosinus des Winkels zwischen diesen beiden Vektoren.

Wenn beide Vektoren vorher normiert wurden, das heißt der Betrag der Vektoren gleich Eins ist, ist die Kosinusdistanz gleich der Skalarproduktdistanz der beiden Vektoren. Außerdem kann in diesem Fall die euklidische Distanz durch folgende Formel aus der Kosinusdistanz errechnet werden. [43]

$\text{euclidean\_distance} = \sqrt{2-2\cos(\theta)}$

3.3 Transformer-Architektur

3.3.1 Sequenz-zu-Sequenz-Modelle

Eine Transformerarchitektur ist eine der modernsten und leistungsfähigsten Architekturen, um eine Vielzahl von NLP-Aufgaben zu lösen [68]. Sie bildet dabei den Nachfolger bzw. Konkurrenten zu den bis dato vorherrschenden rekurrenten neuronalen Netzen (RNN), Gated Recurrent Units (GRU) oder Long-Short Term Memory Systems (LSTM) [33]. Ähnlich zu diesen Architekturen ist auch der Transformer ein Sequenz-zu-Sequenz-Modell. Der Encoderteil nimmt als Eingabe eine Sequenz von Tokens (einen Satz, eine Audiodatei) und der Decoderteil bildet daraus eine andere Sequenz von Tokens (einen Satz, eine Audiodatei).

Der Unterschied von Sequenz-zu-Sequenz-Modellen zu einem gewöhnlichen feedforward neuronalen Netz besteht darin, dass die Eingabe- und Ausgabesignale auch Sequenzen mit variabler Länge sein können. Gewöhnliche künstliche neuronale Netze haben eine feste Anzahl an Eingangs- und Ausgangsneuronen, was es ihnen nur erlaubt, Aufgaben zu lösen, bei denen die Eingangs- und Ausgangsdaten eine bestimmte Länge aufweisen. Anwendungsfälle für künstliche neuronale Netze sind zum Beispiel Frühwarnsysteme oder Klassifikationsaufgaben (z.B. Ziffererkennung) [19] [48].

3.3.2 RNNs, LSTMs und GRUs

Rekurrente neuronale Netze (RNNs) [14] stellen eine Weiterentwicklung von feedforward neuronalen Netzen dar, indem sie eine Rückkoppelung des Ausgabesignals erlauben und somit eine variable Länge an Eingabesignalen verarbeiten können. Das Eingabesignal muss einer Sequenz entsprechen und wird dafür in Tokens zerlegt. Jedes Token wird dabei einmal durch das RNN propagiert, bis ein Ausgabeembedding für dieses Token produziert wurde. Das Ausgabeembedding aus der vorherigen Iteration wird dann zusammen mit dem nächsten Token in der Sequenz als Eingabe für die nächste Iteration verwendet. Dadurch wird die gesamte Sequenz nacheinander in diesem Netz verarbeitet, wobei jedes Token den Kontext der vorherigen Tokens als Embedding mitgeliefert bekommt. Am Ende liefert ein RNN dann ein einziges Embedding, welches die gesamten Informationen der Sequenz repräsentieren soll.

Ein Problem, welches diese Architekturen besitzen, ist das Problem des verschwindenden Gradienten [5]. RNNs versuchen immer im Encoder, die gesamte Inputsequenz von links nach rechts (forward) oder von rechts nach links (backward) zu verarbeiten, um daraus ein einzelnes Embedding zu erstellen. Informationen, die in der Mitte der Sequenz vorkommen, werden wahrscheinlich von danach folgenden Informationen überschrieben, weil das Modell versucht, alle Informationen in derselben Matrix zu speichern.

Die Ansätze des Long-Short Term Memory Systems (LSTM) [33] und der Gated Recurrent Unit (GRU) [10] versuchen dieses Problem zu lösen, indem sie Gates einsetzen. Diese Gates bestimmen, welche Informationen aus dem vorherigen Embedding durch das Embedding der nächsten Iteration überschrieben werden dürfen. Dadurch können sie wichtige Informationen länger speichern. Bei einem LSTM werden dafür drei verschiedene Gates benutzt: das Forget Gate, das Input Gate und das Output Gate. Jedes dieser Gates hat wiederum eine bestimmte Anzahl an Parametern, die bestimmen, welche Informationen vergessen, welche für den Input der nächsten Iteration benutzt und welche bei der nächsten Iteration ausgegeben werden. Alle diese Parameter müssen beim Training des LSTMs mittrainiert werden, was zu einem komplizierten Modell und zu einem langsamen Trainingsprozess führt. Bei GRUs gibt es nur ein Update Gate. Sie haben dadurch weniger Parameter, die trainiert werden müssen und vereinfachen damit den Trainingsprozess. [73]

3.3.3 Transformer-Encoder

Der größte Unterschied der Transformerarchitektur zu RNNs, GRUs und LSTMs ist der Attention-Mechanismus. RNNs, GRUs und LSTMs sind darauf angewiesen, den Input sequentiell Token für Token zu verarbeiten, da jede Zelle als Input das Embedding der vorausgehenden Zelle benötigt. Das macht das Training eines Modells sehr zeitaufwendig.

Der große Vorteil der Transformer ist, dass sie parallelisierbar sind. Alle Tokens einer Inputsequenz können synchron verarbeitet werden. Dabei wird jedes Token zunächst durch ein vorher ermitteltes Word Embedding ersetzt. Der Mechanismus des Positional Encodings stellt sicher, dass das Modell die Reihenfolge der Token mitberücksichtigen kann. Dieser Mechanismus berechnet aufgrund der Position des Tokens in der Sequenz ein Positional Embedding auf der Grundlage von Sinus- und Kosinusfunktionen und addiert es mit dem vorherigen Word Embedding. Das resultierende Embedding enthält damit sowohl Informationen über das Token selbst durch das Token Embedding als auch über die Position in der Sequenz durch das Positional Encoding.

3.3.4 Multihead Self-Attention

Transformer setzen Multihead Self-Attention ein. Der positional encodierte Input wird von mehreren Self-Attention-Köpfen verarbeitet, die sogenannte Multihead Self-Attention. Jeder dieser Köpfe spaltet den Input in Query-, Key- und Value-Matrix auf, indem er die ursprüngliche Embeddingmatrix jeweils mit einer vortrainierten Query-, Key- und Valuematrix multipliziert. Die resultierenden Matrizen werden dann mithilfe folgender Formel zur Attention-Matrix umgewandelt:

$$Attention(Q, K, V) = softmax(\frac{QK^t}{\sqrt{d_k}}) * V$$

Der Parameter d stellt dabei die Dimension der Matrix k dar.

Um die Formel besser zu verstehen, wird an einem Beispiel demonstriert, was die Aufgaben der einzelnen Matrizen sind. Als Eingabe wird der Satz: „Alice fuhr gestern durch die Nürnberger Innenstadt“ benutzt. Die Query-Matrix beschreibt nun eine Frage an diesen Satz, zum Beispiel „Wer?“. Wie genau diese Fragen aussehen, ist nicht leicht zu ermitteln und wird nur in seltenen Fällen so ausfallen, wie auch Menschen nach bestimmten Informationen fragen. Wichtig ist jedoch, dass in jedem Attention-Kopf eine bestimmte Frage gestellt wird. Die Key-Matrix hat nun die Antwort darauf, wo diese Informationen im Satz stehen könnten, also wahrscheinlich im ersten Wort. Die beiden Matrizen werden multipliziert, um die Position der Antwort auf die Frage zu erhalten. Das Ergebnis wird über Division mit der Dimension der Matrix skaliert und anschließend mit der Softmax-Funktion normiert, um die Relevanz der Schlüssel anzupassen. Anschließend wird die angepasste Fragen-Schlüssel-Matrix mit der Value-Matrix multipliziert, welche die Informationen der Sequenz beinhaltet. Die resultierende Matrix würde dann die Information beinhalten, dass die Antwort auf die Frage „Wer?“ für den Satz „Alice fuhr gestern durch die Nürnberger Innenstadt“ wahrscheinlich „Alice“ ist. [61]

Die verschiedenen Köpfe der Multihead Self-Attention liefern nun verschiedene Antworten auf verschiedene Fragen und versuchen damit, die Informationen der Sequenz zu codieren. Im Anschluss werden die verschiedenen Matrizen konkateniert und in einem weiteren feedforward neuronalen Netz in ihrer Dimensionalität reduziert. Die finale Embeddingmatrix kann für verschiedene Funktionalitäten eingesetzt werden.

Die Transformerarchitektur benutzt dieses Embedding, um damit im Decoder wieder neue Tokens zu erstellen. Dafür wird eine ähnliche Vorgehensweise wie im Encoder betrieben. Der größte Unterschied zwischen dem Encoder und dem Decoder ist, dass der Encoder den Input nur ein einziges Mal verarbeitet und die daraus entstandene Embeddingmatrix nicht mehr neu berechnet werden muss. Dadurch ist dieser Schritt sehr schnell. Im Gegensatz dazu wird die Decoderphase für jedes einzelne Outputtoken neu durchlaufen. Zunächst wird dabei wieder eine Self-Attention-Matrix über den gesamten bisherigen Ausgabetokens generiert. Die Self-Attention-Matrix wird dann genutzt, um eine weitere Attention-Matrix zu errechnen. Diese benötigt nun wieder eine separate Query-, Key- und Value-Matrix. Im Vergleich zur Self-Attention werden dabei aber nicht alle Matrizen aus derselben Embeddingmatrix errechnet. Nur die Query-Matrix wird aus der vorher generierten Self-Attention-Matrix generiert. Die Key- und die Value-Matrix werden aus der Embedding-Matrix des Encoders berechnet. Dadurch wird die Information aus dem Userprompt bei der Generierung jedes einzelnen Tokens mitberücksichtigt. Die resultierenden Matrizen aller Attentionköpfe werden wieder durch Feedforward-Netze propagiert, um sie zusammenzuführen und die Dimensionalität anzupassen. Am Ende wird die ganze Matrix durch eine Softmax-Funktion geleitet, die die Ausgabewahrscheinlichkeiten für alle Tokens generiert. Das heißt, die Ausgabe ist

ein Vektor, der als Dimensionalität die Länge des Vokabulars der Tokens besitzt. Welches Token am Ende generiert wird, kann zum Beispiel mit einem Greedy-Algorithmus bestimmt werden, der immer das Wort mit der größten Wahrscheinlichkeit auswählt. Andere Auswahlmöglichkeiten wären zum Beispiel mit einem Beam-Search-Algorithmus möglich, bei dem mehrere Sequenzen generiert werden, um erst später zu schauen, welche Sequenz insgesamt die höchste Wahrscheinlichkeit besitzt. In dieser Phase kann auch eine Zufallsvariable eingeführt werden, die immer ein Token aus den wahrscheinlichsten Token auswählt, um bei mehrfacher Wiederholung eines Prompts verschiedene Ergebnisse zu liefern. [87]

Bei dem Beispiel von oben würde der Decoder ggf. versuchen, den Satz zu vollenden mit „Alice fuhr gestern durch die Nürnberger Innenstadt und bewunderte die beeindruckenden mittelalterlichen Bauwerke.“

3.4 Large Language Model

Large Language Model (LLM) bezeichnet ein Sprachmodell, das auf vielen Parametern trainiert wurde. Viele modernen LLMs basieren auf der Transformerarchitektur [68], um aus einer Inputsequenz mithilfe des Encoders und des Decoders wieder eine Outputsequenz zu generieren.

Technisch gesehen ist die Funktionsweise eines LLMs nur eine Next-Token-Prediction. Dabei wird nach Verarbeitung der Inputsequenz eine Wahrscheinlichkeit des Auftretens verschiedener Token ermittelt. Das LLM generiert so einfach nur Sätze, die in dem vorherigen Kontext Sinn ergeben. Durch weitere Techniken des Finetunings können die LLMs dazu gebracht werden, auf Fragen zu antworten oder Anweisungen zu befolgen. Die LLMs werden mit weiteren Daten trainiert, in denen zum Beispiel das Antworten auf Fragen explizit demonstriert wird [66]. Verschiedene LLMs können dabei auf unterschiedliche Aufgaben spezialisiert werden. LLMs werden zum Beispiel für Chatbots, Text-Übersetzungen, Sprach-Übersetzungen, Named Entity Recognition, Sentimentanalyse oder Klassifikation eingesetzt [68].

3.5 Chatbots

Besonders die LLM-gestützten Chatbots werden aufgrund ihrer Leistungsfähigkeit immer beliebter. Der bekannteste Chatbot ist ChatGPT von OpenAI [8], welcher über 100 Millionen Nutzer und über eine Milliarde Aufrufe im Monat hat [85]. Weitere bekannte Chatbots sind Google Gemini [28], die Claude-Chatbots [13] von der Firma Anthropic oder Pi von Inflection AI [71].

Neben den bekannten kommerziellen Chatbots, die von großen Tech-Firmen entwickelt werden, entstehen immer mehr Open-Source-Modelle, die zwar noch nicht so leistungsstark sind, aber auf eigener Hardware lokal laufen können. Bekannte Open-Source-Modelle sind zum Beispiel Gemma von Google [4], LLama 2 von Meta [86] oder Mixtral von Mistral AI [39]. Diese Open-Source-Modelle sind meist wesentlich kleiner als die kommerziellen Chatbots und sind somit auf lokaler Consumer-Hardware ausführbar. Typische Größen für Open-Source-LLMs sind 7 Milliarden-, 13 Milliarden-, 30 Milliarden- und 70 Milliarden-Parameter [86], das neueste Google-Gemma-Modell hat sogar eine Variante mit nur zwei Milliarden-Parametern [4].

Die Vorteile eines selbst gehosteten LLMs sind vor allem die Datensicherheit und die Möglichkeit, die Modelle auf die eigenen Aufgaben zu spezialisieren durch die Technik des Fine-Tunings [66]. Dabei werden Basis-Chatbots wie LLama 2 auf weiteren spezifischen Beispielen trainiert, um bestimmte Fähigkeiten des Modells zu verbessern.

In dieser Arbeit wird aufgrund von fehlenden konsistenten Hardwareressourcen und zugunsten der Schnelligkeit noch auf das kommerzielle Produkt ChatGPT [8] zugegriffen, um Datenverarbeitungen durchzuführen. In Zukunft könnte aufgrund der immer besseren und kleineren Open-Source-LLMs diese Funktionalität auch auf selbstgehostete LLMs verlagert werden.

3.6 Hugging Face

Da es mittlerweile eine Vielzahl unterschiedlicher LLMs gibt, die alle verschiedene spezielle Aufgaben besitzen und in bestimmten Aspekten jeweils Vor- und Nachteile aufweisen, ist es sinnvoll, diese untereinander vergleichen zu können. Eine solche Vergleichsmöglichkeit bietet die Plattform Hugging Face [88], auf der über 500.000 Machine-Learning-Modelle, davon mehr als 200.000 Transformer, gehostet werden.

In der Kategorie „Question Answering“ gibt es über 1300 Einträge und in der Kategorie „Sentence Similarity“ ca. 3000 [55]. In beiden Kategorien existieren Embedding-Modelle. Die einzelnen Modelle können kostenlos heruntergeladen werden, sie sind ausführlich dokumentiert und sie sind untereinander mithilfe einer Punkteabgabe auf verschiedenen Kontroll-Datensätzen vergleichbar. Durch Leaderboards können die besten Modelle bestimmt werden.

3.7 Massive Text Embedding Benchmark

Hier wird beispielhaft das Massive Text Embedding Benchmark (MTEB) [58] Leaderboard betrachtet.

Auf diesem Leaderboard werden Embedding-Modelle auf 58 verschiedenen Datenbeständen in 112 Sprachen evaluiert [58]. Die Datenbestände sind dabei in acht verschiedene Aufgabenkategorien geteilt: Bitext Mining, Classification, Clustering, Pair Classification, Reranking, Retrieval, Semantic Textual Similarity und Summarization.

Für diese Arbeit sind vor allem die einzelnen Aufgabenbereiche Retrieval und Reranking interessant. Im Bereich Retrieval besteht die Aufgabe, aus einer großen Menge an Daten, relevante Dokumente zu einem bestimmten Thema zu finden. Bei der Reranking-Aufgabe gilt es, eine Liste an potenziell passenden Dokumenten nach Wichtigkeit zu sortieren. Im Aufgabenbereich Retrieval werden die einzelnen Embeddings auf 15 verschiedene Datenbeständen evaluiert. [58]

Leider ist eine Evaluation auf deutscher Sprache noch nicht möglich, da zu wenig Datenbestände zur Verfügung stehen. In Zukunft, wenn mehr Datenbestände für die deutsche Sprache entwickelt wurden, wird die Evaluation um eine deutsche Kategorie erweitert. [57] In Kapitel 6 wird versucht, die besten Embedding-Modelle für die Retrieval-Aufgabe des Podcast-Generators zu ermitteln.

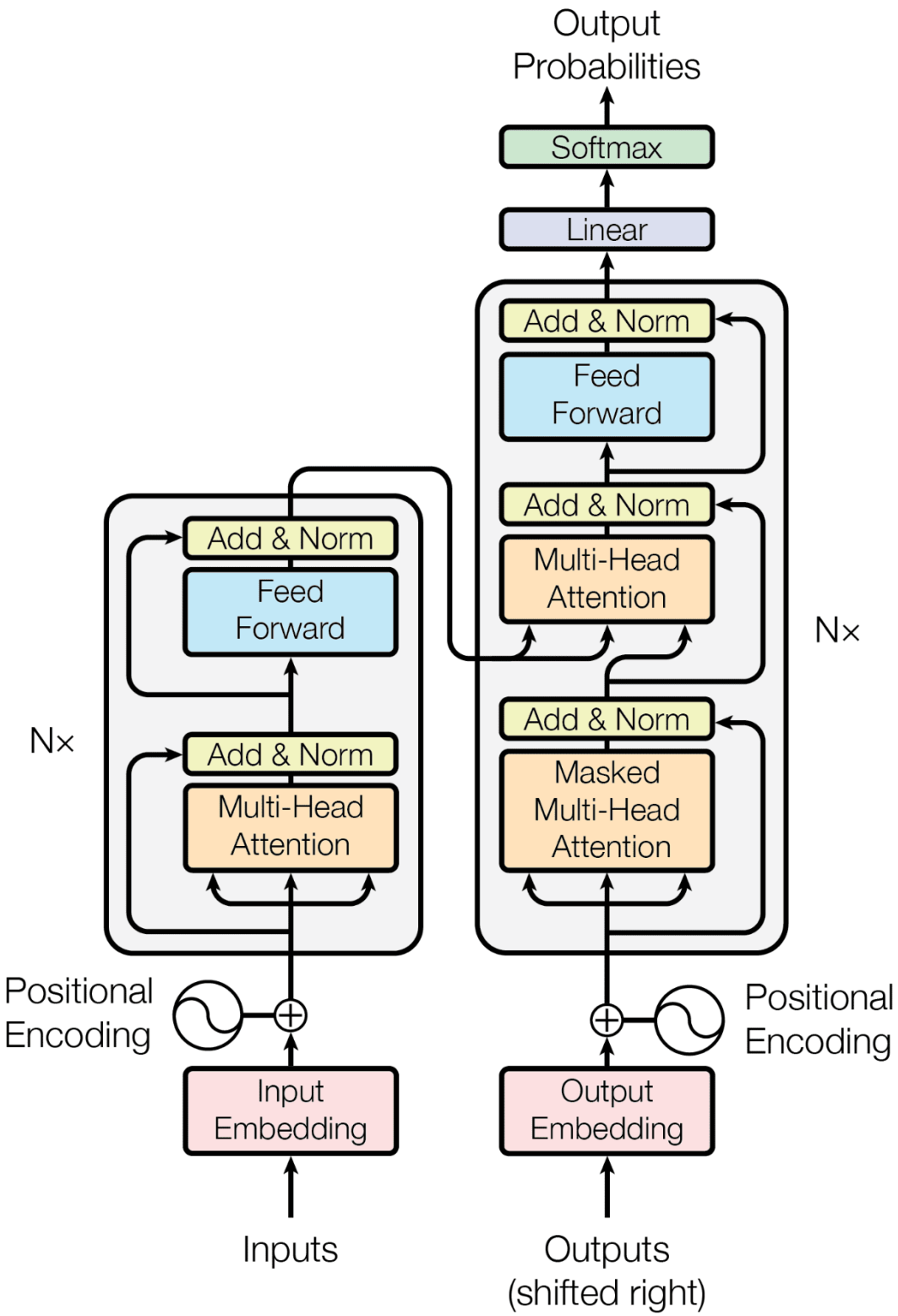


Abbildung 3.1: Transformer Architektur aus [87]

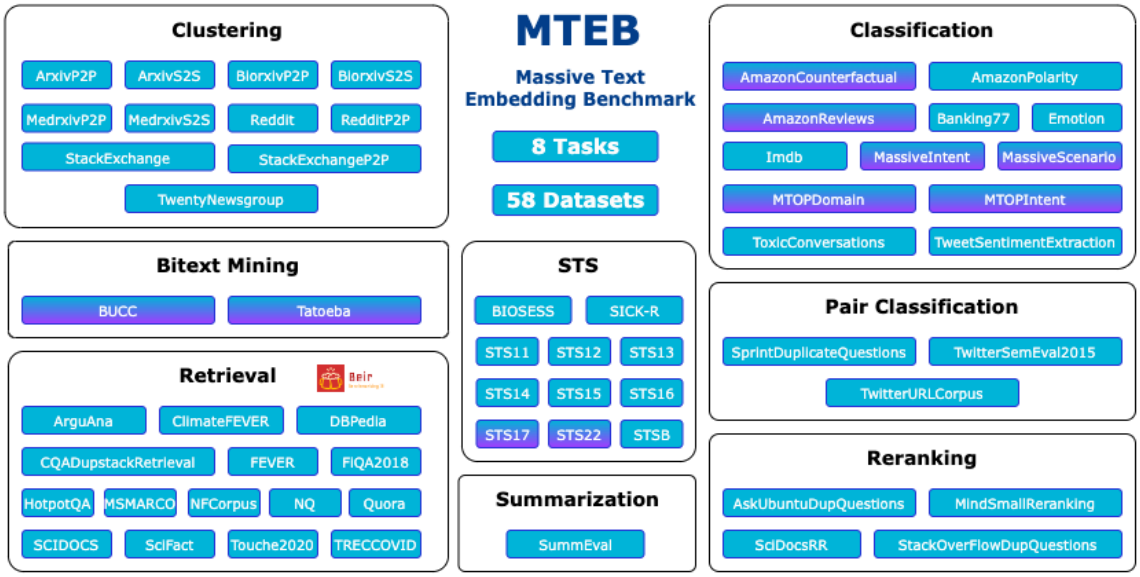
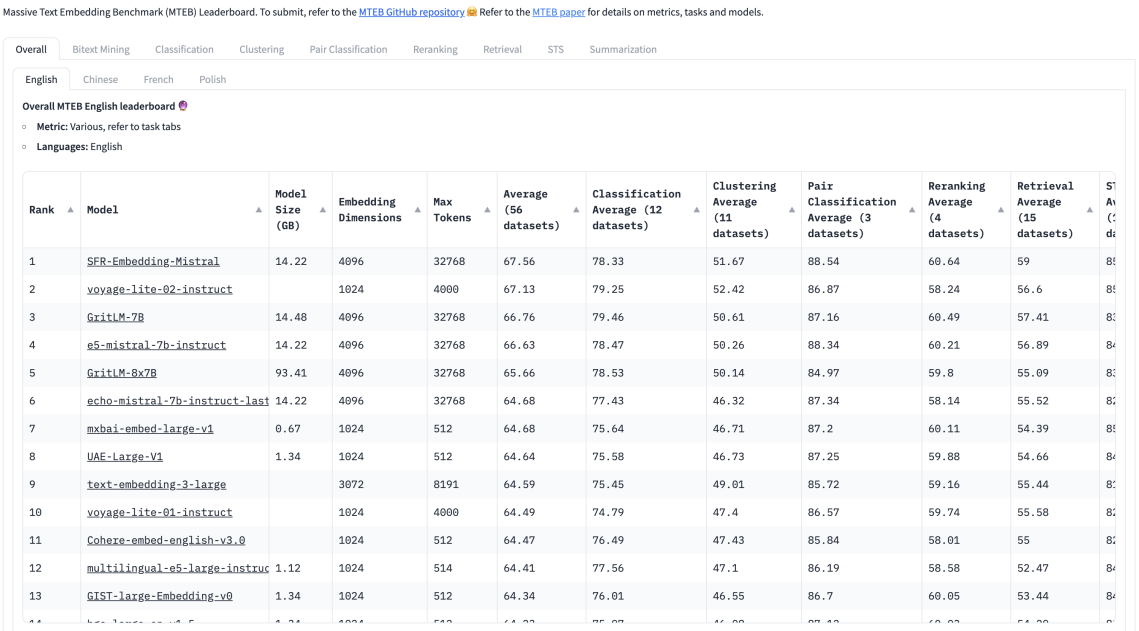


Abbildung 3.2: Die einzelnen Datenbestände in MTEB aus [58]



Massive Text Embedding Benchmark (MTEB) Leaderboard. To submit, refer to the MTEB GitHub repository Refer to the MTEB paper for details on metrics, tasks and models.

Overall | Bitext Mining | Classification | Clustering | Pair Classification | Reranking | Retrieval | STS | Summarization

English | Chinese | French | Polish

Overall MTEB English leaderboard

- **Metric:** Various, refer to task tabs
- **Languages:** English

Rank	Model	Model Size (GB)	Embedding Dimensions	Max Tokens	Average (56 datasets)	Classification Average (12 datasets)	Clustering Average (11 datasets)	Pair Classification Average (3 datasets)	Reranking Average (4 datasets)	Retrieval Average (15 datasets)	S… A… (… da…
1	SFR-Embedding-Mistral	14.22	4096	32768	67.56	78.33	51.67	88.54	60.64	59	85
2	voyage-lite-02-instruct		1024	4000	67.13	79.25	52.42	86.87	58.24	56.6	85
3	GritLM-7B	14.48	4096	32768	66.76	79.46	50.61	87.16	60.49	57.41	83
4	e5-mistral-7b-instruct	14.22	4096	32768	66.63	78.47	50.26	88.34	60.21	56.89	84
5	GritLM-8x7B	93.41	4096	32768	65.66	78.53	50.14	84.97	59.8	55.09	83
6	echo-mistral-7b-instruct-last	14.22	4096	32768	64.68	77.43	46.32	87.34	58.14	55.52	82
7	mxbai-embed-large-v1	0.67	1024	512	64.68	75.64	46.71	87.2	60.11	54.39	85
8	UAE-Large-V1	1.34	1024	512	64.64	75.58	46.73	87.25	59.88	54.66	84
9	text-embedding-3-large		3072	8191	64.59	75.45	49.01	85.72	59.16	55.44	83
10	voyage-lite-01-instruct		1024	4000	64.49	74.79	47.4	86.57	59.74	55.58	82
11	Cohere-embed-english-v3.0		1024	512	64.47	76.49	47.43	85.84	58.01	55	82
12	multilingual-e5-large-instruc	1.12	1024	514	64.41	77.56	47.1	86.19	58.58	52.47	84
13	GIST-large-Embedding-v0	1.34	1024	512	64.34	76.01	46.55	86.7	60.05	53.44	84

Abbildung 3.3: Screenshot des MTEB Leaderboards [56]

Kapitel 4

Datenbeschaffung und Datenspeicherung

4.1 Ausgangslage

Um eine Zusammenstellung verschiedener Audiosegmente möglichst genau und inhaltlich abgestimmt auf ein bestimmtes Thema zu fokussieren, müssen auch zu diesem Thema passende Audioabschnitte in den Daten verfügbar sein. Als Audiodaten könnten fast alle Audioressourcen genutzt werden, wie zum Beispiel Beiträge aus Radioprogrammen, die Audiospuren von Videos oder ganze Podcast-Episoden.

Die Qualität dieses Systems würde mit steigender Anzahl an Audiomaterial bessere Ergebnisse erzielen, da dann zu vielen Themen mehr Inhalte zur Verfügung stehen. Allerdings ist es im Umfang dieser Arbeit nicht möglich, alle verfügbaren Audiomaterialien für die Erstellung der Podcasts zu verwenden. Die Audiodaten müssten dazu erst transkribiert werden und anschließend mit diesen Transkripten Embeddings generiert werden, was zeit- und ressourcenintensiv ist. Für diese Arbeit wurde versucht, möglichst qualitativ hochwertige Daten zu benutzen, um mit den verfügbaren Ressourcen und in der gegebenen Zeit einen qualitativ hochwertigen Prototypen zu erstellen.

Als Audiomaterial würden sich am besten Podcasts eignen, in denen über verschiedene Themen sachlich gesprochen wird, die Informationen aber gleichzeitig faktisch korrekt sind. Außerdem sollten die Audioquellen frei verfügbar sein, damit keine Urheberrechtsverletzung stattfindet.

4.2 Beschaffung der Audiodaten

4.2.1 Die ARD-Audiothek

Die ARD-Audiothek ist in Deutschland eine der größten Audio- und Podcastanbieter mit mittlerweile über 100.000 verschiedenen verfügbaren Audioinhalten und über 116 Millionen

Audioabrufen allein im Jahr 2023 [3]. Alle Inhalte unterliegen den journalistischen Grundsätzen der ARD und bieten somit einen sorgfältigen Qualitätsstandard [41]. Die verschiedenen Audioinhalte stammen von den einzelnen Landesrundfunkanstalten, der ARD und dem Deutschlandradio und liefern eine Vielzahl verschiedener Inhalte. Sie enthält über 2000 verschiedene Podcasts in vielen unterschiedlichen Kategorien, wie Comedy, Sport, Wissenschaft, Wirtschaft, Gesellschaft, Kunst, Musik oder Philosophie. In dieser Audiothek gibt es zudem Hörbücher, Hörspiele oder Podcasts nur für Kinder. Die einzelnen Rundfunkanstalten tragen außerdem eigene Podcasts bei, die meist einen regionalen Kontext haben, wie zum Beispiel der Podcast „Giga Grünheide“ über das Tesla-Werk in Brandenburg vom rbb [29]. Alle diese Inhalte sind für nicht-kommerzielle Forschungsprojekte kostenlos und frei verfügbar und stellen eine gute Quelle für das Audiomaterial dar, das in dieser Arbeit verwendet wird [63].

4.2.2 Podcastreihe radioWissen

Zur automatischen Generierung von Podcast-Episoden bietet es sich an, dass in den Ausgangsaudios die Sprache klar und verständlich ist, sowie verschiedene Sprecher sich nicht ins Wort fallen bzw. gleichzeitig reden. Außerdem ist es wünschenswert, die ausgeschnittenen Audiosegmente an klaren Satzgrenzen zu teilen, sodass der Ausschnitt nicht mitten im Satz beginnt und den Zuhörenden der Kontext vorenthalten wird.

Aufgrund dieser Kriterien wurde als Datengrundlage die Podcastreihe radioWissen von Bayern 2 [76] benutzt. Diese ist nicht wie ein klassischer Podcast im Dialogstil aufgebaut, sondern ähnelt einem Hörspiel, bei dem der Text von einem Manuskript abgelesen wird. Dazu kommen verschiedene Geräusche und Stimmen, um den Hörenden mehr Abwechslung zu bieten.

Der Fokus der einzelnen Episoden liegt auf interessanten Beiträgen zu verschiedenen Themen, die häufig Gebiete der Geschichte, Naturwissenschaft, Gesellschaft oder Philosophie umfassen. Beispielepisoden sind: „Fasten - Verzicht und innerer Gewinn?“, „Die Maus - Anpassungskünstler und gefürchteter Schädling“, oder „Maria Sibylla Merian - Naturforscherin und Künstlerin“. [76]

Die mehr als 2000 Episoden wurden von mehr als 150 verschiedenen Autoren geschrieben [76]. Dadurch sind die einzelnen Episoden unterschiedlich in ihrer Erzählweise. In einigen Episoden kommen originale Audiospuren von historischen Aufnahmen vor oder auch Gastbeiträge von Experten. Zudem wird beinahe jeder Podcast abwechselnd von mehreren Stimmen vorgetragen, was nachweislich die Aufmerksamkeit von Zuhörenden verbessert [42].

4.2.3 Datenbeschaffung über die ARD-Audiothek-API

Die Inhalte der ARD-Audiothek können entweder direkt über die Webseite erreicht oder mithilfe einer frei benutzbaren Web-GraphQL-API abgefragt werden [74]. Über diese Schnittstelle sind alle Informationen, wie Titel, Beschreibungen, Autoren oder auch der Link zum MP3-File jeder Episode abrufbar.

Zunächst müssen alle Downloadlinks zu den einzelnen Episoden ermittelt werden. Mit der GraphQL-Abfrage (s. Kapitel 8) werden alle Download-Links zu den Podcast-Episoden des Podcasts „radioWissen“ von Bayern 2 ermittelt. Das sind (Stand 3. Januar 2024) 2257 Podcast-Episoden.

Alle diese Audiodateien wurden anschließend heruntergeladen und auf einer lokalen Festplatte gespeichert.

Für weitere Analysen und die Kategorisierung der Audiodateien ist es außerdem sinnvoll, die Beschreibungen der einzelnen Episoden und die dazugehörigen Schlagwörter abzufragen, da diese eine kurze Zusammenfassung oder Einordnung der Episoden enthalten. Außerdem bietet das Entstehungsdatum der Episoden die Möglichkeit, Informationen später nach Aktualität zu filtern. Über die Anfrage (s. Abschnitt 8) können all diese Informationen abgefragt werden.

Über die API kann auch in einigen Fällen direkt ein Transkript des Audiofiles angefordert werden. Allerdings ist die Transkription meist nicht sehr akkurat. Näheres dazu in Kapitel 5.

4.2.4 Audiodatenanalyse

Die 2232 einzigartigen Episoden haben eine durchschnittliche Länge von 22 Minuten und eine durchschnittliche Größe von 21 MB. Insgesamt weisen diese Audiodaten eine Größe von ungefähr 47 GB auf.

Dabei kam es insgesamt 15-mal vor, dass zwei Episoden denselben Titel tragen, aber eine unterschiedliche Download-URL aufwiesen. Die Download-URLs unterscheiden sich nur, indem am Ende die Zeichen „-1“ oder „-2“ angefügt wurden. Zum Beispiel endet der Download-Link der Episode „Quantenphysik - Wahr, aber verrückt“ auf `quantenphysik-wahr-aber-verrueckt.mp3` aber auch auf `quantenphysik-wahr-aber-verrueckt-1.mp3`. In diesem Fall liefert nur die zweite URL einen Download, die erste zeigt eine Fehlermeldung an. Es gibt auch Fälle, in denen beide Links funktionieren, wie zum Beispiel `die-bamberger-hexenprozesse-unschuldig-muss-ich-sterben.mp3` und `die-bamberger-hexenprozesse-unschuldig-muss-ich-sterben-1.mp3`. Die Daten wurden dementsprechend bereinigt und doppelte Audioinhalte nur einmal abgespeichert.

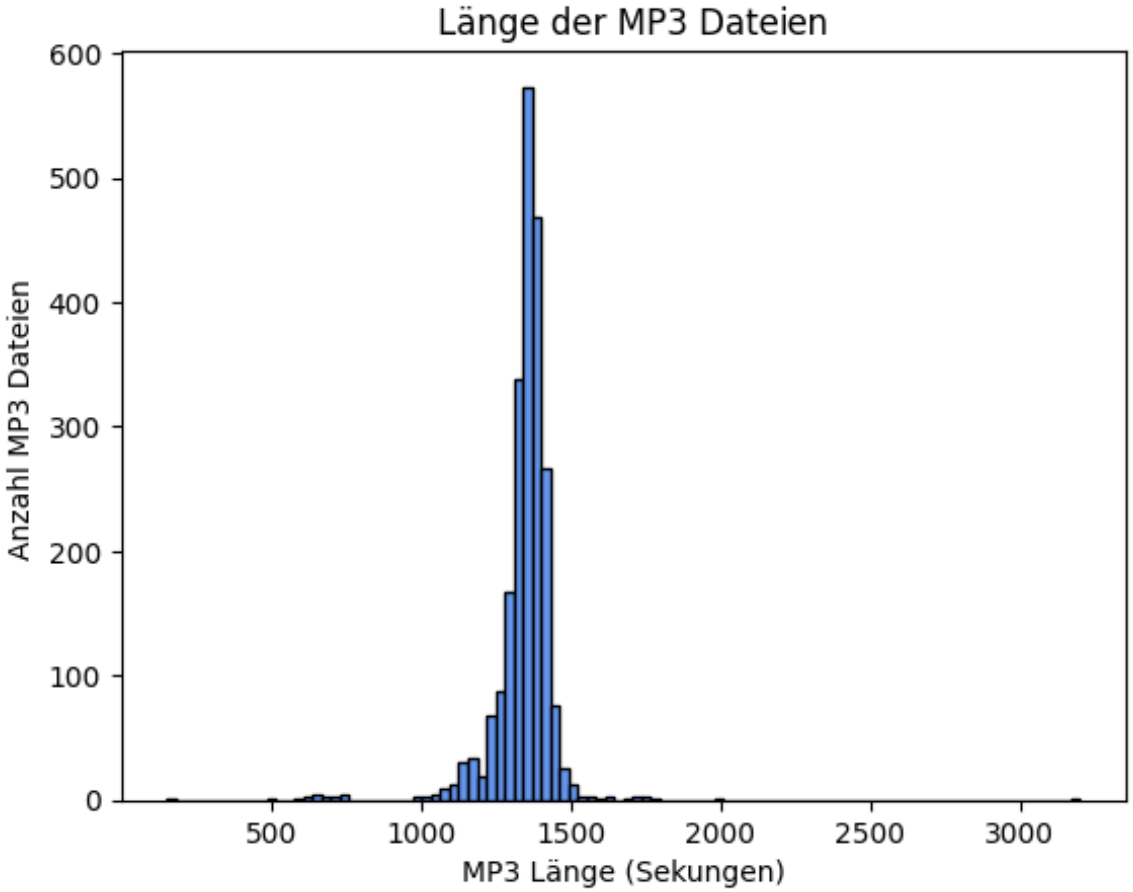


Abbildung 4.1: Länge der MP3 Dateien

4.3 Transkription der Podcasts (ASR)

4.4 Vorgehensweise

Zunächst müssen die Daten gesammelt und aufbereitet werden. Für dieses Projekt bildet die Datengrundlage die Transkripte bzw. Manuskripte der Podcasts der ARD-Audiothek. In der Audiothek selbst gibt es keine Transkripte zu den Podcasts. Für den Podcast „radioWissen“ von Bayern 2 gibt es auf deren Seite die Manuskripte im PDF-Format [78]. Diese sind zwar inhaltlich hochqualitativ, besitzen aber keine Zeitinformationen zu den einzelnen Wörtern. Die Zeitinformationen in Form von Zeitstempeln für jedes Wort sind wichtig, um die Audiofiles der Podcasts später an den richtigen Stellen zuzuschneiden.

Ein anderer Ansatz ergibt sich, wenn die Podcasts transkribiert werden. Die Vorteile sind, dass die Transkription auch bei Podcasts funktioniert, für die vorab kein Transkript erstellt wurde, was die Mehrzahl aller Podcasts ausmacht. Außerdem kann bei einer Transkription auch gleichzeitig die Zeitstempel für jedes Wort extrahiert werden.

4.4.1 Transkriptionsmethoden im Überblick

Um automatisch Audioinhalte mittels eines Automatic Speech Recognition Systems (ASR) zu transkribieren, können mittlerweile zahlreiche verschiedene Dienste benutzt werden. Dabei muss immer zwischen Geschwindigkeit, Genauigkeit und Kosten abgewogen werden. Es gibt viele Cloudanbieter, die die Transkription via API gegen Gebühren vornehmen. Außerdem gibt es verschiedene Transkriptionsprogramme, die auf lokaler Hardware eingesetzt werden können. Im Folgenden sind mehrere Ansätze beschrieben, die für die Transkription der Daten in Erwägung gezogen wurden.

4.4.2 Fraunhofer IAIS

Wie im Kapitel 3 beschrieben, führt das Fraunhofer-Institut für Intelligente Analyse- und Informationssysteme (IAIS) schon automatische Transkriptionen der Audioinhalte der ARD-Audiothek durch. Die Verwendung dieser Transkripte führt allerdings gleich zu mehreren Problemen. Ein Grund ist die inkonsistente Verfügbarkeit der Daten. Von den angefragten Transkripten verfügten 84 Episoden und damit ca. 3,8% aller Episoden über keine Daten für Transkripte. Ein weiterer Grund ist die vorgefertigte Segmentierung des Transkripts. Die Transkripte sind in Segmente aufgeteilt, die zwischen 3-30 Sekunden dauern. Zeitstempel sind jeweils nur für den Anfang und das Ende jedes Segments verfügbar. Dadurch ist eine spätere Segmentierung nicht mehr möglich und die vorhandenen Segmente sind zu inkonsistent in der Länge bzw. beginnen und enden nicht an Satzgrenzen.

4.4.3 Whisper

Eines der besten kostenlosen Open-Source-ASR-Systeme bietet Whisper von OpenAI [75]. Dieses System wurde mit einem maschinellen Lernverfahren auf 680.000 Stunden Audiomaterial in verschiedenen Sprachen trainiert und erreicht damit State-of-the-Art-Performance in Transkriptionen [75]. Es ist sehr leistungsstark und kann lokal auf eigener Hardware laufen.

Konkret wurde in dieser Arbeit das Projekt faster-whisper [83] verwendet, welches die ursprünglichen Whisper-Modelle mithilfe einer schnelleren Inferenz-Engine neu implementiert. faster-whisper bietet außerdem Word-Level-Timestamps im Gegensatz zum ursprünglichen Whisper-Projekt.

Whisper bietet mehrere verschiedene Modelle zur Transkription an. Es gibt die Modelle tiny, base, small, medium und large. Das Basismodell hat ca. 74 Millionen Parameter, benötigt ca. 1 GB VRAM und ist ca. 16-mal schneller als das Large-Modell. Bei einem Test für die Episode „1968 - Das Ausnahmejahr“ [1] aus dem Podcast radioWissen von Bayern 2

schneidet es aber nicht sehr gut ab. Aus dem Wort „Vietnam“ wird „Wirdnam“, aus „Panzer in Prag“ wird „Panzer-Inprac“ und aus „Ohrfeige“ wird „Urfeige“. Der vorgetragene Text wurde dabei ohne Störgeräusche und von einer Person flüssig vorgetragen. Eine Evaluation der Transkriptionsfähigkeiten mittels einer Word Error Rate (WER) ist nicht durchgeführt worden, da notwendige Vergleichstranskripte nicht vorhanden sind.

Dagegen bietet das Medium-Modell von Whisper deutlich bessere Ergebnisse für dieselbe Episode. Bei der Transkription konnte kein Fehler festgestellt werden. Allerdings ist der Zeitaufwand durch höhere Rechenleistung immens. Auf einem MacBook Pro 2016 mit einem Intel Core i7 benötigt die Transkription ca. 45 Minuten pro Episode. Auf einer T4-GPU, wie sie Google kostenlos auf Google Colab zur Verfügung stellt, dauert eine Transkription immer noch 3,5 Minuten. Für ca. 2000 Episoden benötigt diese Methode demnach ca. 7000 Minuten (ca. 116 Stunden).

Die Transkription für diese Arbeit wurde auf einem High-Performance-Cluster der Technischen Hochschule Nürnberg durchgeführt. Dabei wurde eine NVIDIA A100-GPU verwendet. Über vier separate Slurm-Jobs wurden die 2237 Episoden innerhalb von ca. 50 Stunden transkribiert.

Die Transkripte der Episoden sind im Durchschnitt ca. 2792 Wörter lang und benötigen ungefähr 20 KB Speicherplatz pro Transkript.

4.4.4 Eurovox

Für die zukünftige Verwendung dieses Systems im öffentlich-rechtlichen Kontext würde es sich anbieten, weitere Podcast-Episoden mittels Cloud-Computing zu transkribieren. Das heißt, das Whisper-Modell nicht auf der lokalen Hardware laufen zu lassen, sondern zum Beispiel auf den Servern von Eurovox. Eurovox ist ein Softwaretool von der EBU, der European Broadcasting Union. Sie ist ein Zusammenschluss von derzeit 68 Rundfunkanstalten in 56 Staaten Europas, Nordafrikas und Vorderasiens mit Sitz in Genf. Das Tool Eurovox steht dabei allen Mitgliedern zur Verfügung. Mithilfe dieses Tools können Text-to-Speech, Übersetzungen und Speech-to-Text Services über ein UI, oder eine API benutzt werden. Für dieses Projekt könnte zunächst die Text-to-Speech-Funktion verwendet werden. Leider wurde der Zugang zu der API für dieses Projekt noch nicht freigestellt. [23]

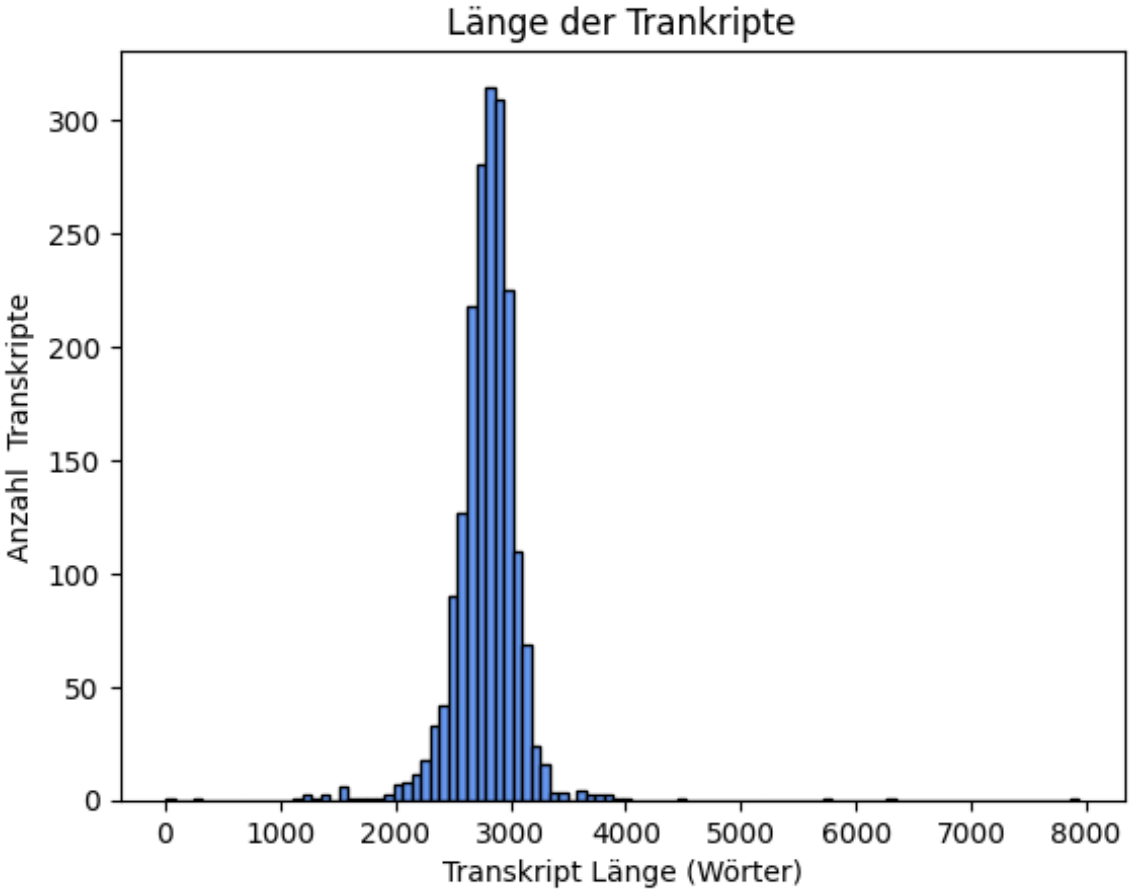


Abbildung 4.2: Anzahl der Wörter pro Transkript

4.5 Datenaufbereitung

4.5.1 Segmentbildung

Die von Whisper transkribierten Daten enthalten die erkannten Wörter sowie die einzelnen Zeitstempel für den Start und das Ende jedes Wortes mit einer Genauigkeit von einer Zehntelsekunde. Für die spätere Analyse der Daten ist es sinnvoll, die einzelnen Wörter in größere Segmente zusammenzufassen, damit bei einer Suchfunktion der Kontext von umliegenden Wörtern miteinbezogen werden kann.

Für die Wahl der richtigen Segmentgröße gibt es keine einheitliche Lösung. Die Wahl hängt unter anderem von der Strukturierung des zu embeddenden Inhaltes, den Fähigkeiten des Embedding-Modells, sowie der Länge und Komplexität der einzelnen User-Queries ab.

Der primitivste Ansatz wäre, die Transkripte einfach in Blöcke einer bestimmten Größe aufzuteilen. Dann wäre jedes Segment zum Beispiel 50 Wörter lang. Der offensichtliche Nachteil eines solchen Ansatzes ist, dass die Segmente keinen Bezug zu dem Inhalt der

Transkripte besitzen. Die resultierenden Segmente würden mitten im Satz anfangen und aufhören und dadurch auch Informationen schlecht repräsentieren. Zusätzlich würden diese Segmente bei der Rückübersetzung in Audio eine schlechte Nutzererfahrung bieten, da auch die Audiosegmente dann mitten im Satz anfangen und enden würden.

Ein etwas organisierterer Ansatz besteht darin, die einzelnen Wörter zunächst in Sätze zu gruppieren. Für diese Gruppierung werden zwei Ansätze betrachtet. Der erste Ansatz besteht darin, die von Whisper erzeugten Satzpunkte als Trennzeichen für einen Satz zu erlauben. Als zweiter Ansatz wird eine tiefergehende Analyse der Transkripte mithilfe der Sprachbibliothek spaCy durchgeführt.

4.5.2 Satzbildung mit Whisper

Whisper erkennt von sich aus, ob ein Wort das Ende eines Satzes markiert. Das geschieht wahrscheinlich vor allem aufgrund der Tonlage und der Pause zwischen Wörtern und der grammatikalischen Struktur der vorangegangenen Wörter [6] [75]. Dann gibt Whisper das Wort mit einem Punkt, einem Fragezeichen oder einem Ausrufezeichen am Ende als erkanntes Wort aus. Diese Interpunktion kann benutzt werden, um die einzelnen Wörter in Sätze zu gruppieren. Leider ergibt sich dabei das Problem, dass bestimmte Wörter oder Abkürzungen zusätzliche Punkte enthalten. Beispiele sind „Mr. Smith“, „seinem 26. Studioalbum“, „am 10. Januar 2016“. Mit diesem naiven Ansatz der Satztrennung würden einige Sätze an ungewollten Stellen in mehrere Sätze aufgetrennt werden. Die resultierenden Sätze bieten weniger inhaltlichen Zusammenhang und sind meist grammatikalisch nicht vollständig, was bei einer Rücküberführung in Audio eine schlechtere Nutzererfahrung bietet.

4.5.3 Satzbildung mit spaCy

Um dieses Problem zu lösen, muss ein Algorithmus verstehen, welche Satzzeichen die wirklichen Satzenden anzeigen. Dies erfordert ein Verständnis der Satzstruktur und ist deswegen nicht trivial möglich. Um die Satzgrenzen zu finden, wird deshalb die Open-Source-Sprachbibliothek spaCy verwendet [35].

Mithilfe der NLP-Bibliothek spaCy kann ermittelt werden, welche Satzzeichen wirklich die Grenze eines Satzes markieren. spaCy verwendet dafür Machine-Learning-Modelle, die aufgrund von vielen Daten gelernt haben, wo das Ende eines Satzes ist. Auf der offiziellen Webseite von spaCy werden vier verschiedene Modelle für die deutsche Sprache zur Verfügung gestellt. Das Modell `„de_core_news_sm"` ist das kleinste Modell mit 13 MB Größe, dann folgt `„de_core_news_md"` mit einer Größe von 42 MB und `„de_core_news_lg"` ist

das größte Modell mit 541 MB. Das größte Modell bietet dabei vor allem viele vortrainierte Embedding-Vektoren für einzelne Wörter. Das vierte Modell `„de_dep_news_trf"` verbraucht 391 MB Speicherplatz und benutzt eine Transformer-Architektur. [81]

spaCy führt eine selbst angegebene Accuracy Evaluation für Sentence Segmentation, also das Erkennen von Satzenden in einem Text, durch. Darin ist der F-Score für das kleine Modell gleich 0,94; für das mittlere Modell 0,95 und für das große Modell ebenfalls 0,95. Das Modell mit der Transformer-Pipeline bietet einen F-Score von 0,98 und wird deswegen in dieser Arbeit verwendet. [81]

Mithilfe dieses Ansatzes wird der folgende Satz beispielsweise richtig erkannt: „Zwei Tage nach seinem 69. Geburtstag und der Veröffentlichung von Blackstar, seinem 26. Studioalbum."

4.5.4 Segmente anpassen

Da wichtige Informationen über mehrere Sätze verteilt liegen können, kann es bei einfacher Segmentierung in Sätzen zu dem Problem kommen, dass Informationen über mehrere Sätze verteilt sind. Oft ist das zum Beispiel bei Pronomen der Fall, wenn diese sich auf ein Nomen in einem vorherigen Satz beziehen. Es gibt mehrere Möglichkeiten, dieses Problem anzugehen, zum Beispiel, indem Koreferenzauflösung [47] durchgeführt wird und dadurch Pronomen durch ihre eigentlichen Substantive ersetzt werden. In einem weiteren Ansatz könnten Segmente aus mehreren Sätzen gewählt werden, die sich gegenseitig überlappen. Dann würde jedes Segment auch den Kontext von mehreren Sätzen darstellen und die Informationen über Satzgrenzen hinaus erfassen. Beide Ansätze wurden in dieser Arbeit nicht verfolgt, bieten aber Möglichkeiten für zukünftige Weiterentwicklungen.

4.5.5 Segmente für TF-IDF vorbereiten

4.5.5.1 Lemmatisierung

Für die Suche mithilfe des TF-IDF-Algorithmus ist es nützlich, die Segmente vorher zu bearbeiten. Standardmäßig nutzt der TF-IDF-Algorithmus ganze Wörter als Tokens. Durch effektive Bearbeitungsschritte können jedoch mehrere ähnliche Wörter zu einem Stammwort reduziert werden. Bei der Suche nach dem Wort „Wandern" sollen zum Beispiel auch Segmente gefunden werden, die die Wörter „Wanderung" oder „Wanderer" enthalten. Um diese Abbildung von mehreren Wörtern auf ein Stammwort zu erreichen, gibt es zwei unterschiedliche Methoden: Stemming und Lemmatisierung.

Stemming versucht regelbasiert, Suffixe von Wörtern zu entfernen, um aus Wörtern mit verschiedenen Endungen, wie zum Beispiel „Wanderer" und „Wanderung", das Stammwort

„Wander“ zu bilden. Der Vorteil ist, dass der Algorithmus sehr schnell agieren kann, um die Wörter auf ihre Stammform zu reduzieren. Die Nachteile sind jedoch, dass die deutsche Sprache sehr viele verschiedene Wortkonstrukte zulässt, die nicht alle mithilfe von einzelnen Regeln auf ein gemeinsames Stammwort gebracht werden können, zum Beispiel der Plural „Bäume“ von „Baum“.

Um auch solche Wörter auf ihre Stammwörter zu reduzieren, kann eine Lemmatisierung erfolgen. Bei der Lemmatisierung gibt es eine vordefinierte Tabelle, die für jedes Wort einen Eintrag zu seinem Stammwort besitzt. Einen solchen Lemmatisierer bietet zum Beispiel die Sprachbibliothek spaCy. Dafür wird das Modell `„de_core_news_md"` verwendet, da es bei der Accuracy Evaluation einen F-Score von 0,98 erreicht und damit ebenso gut ist wie das `„de_core_news_lg"` und besser als `„de_core_news_sm"`. [81] Das Vokabular der von Whisper transkribierten Podcast-Episoden umfasst 235.528 verschiedene Wörter. Mithilfe des Lemmatisierungsprozesses schrumpft das Vokabular um 15% auf 200.341.

4.5.5.2 Kompositatrennung mit german compound splitter

Als weiteren Vorverarbeitungsschritt für die effiziente Keywordsuche kann man zusammengesetzte Wörter in ihre Bestandteile auftrennen. Das Vokabular des Datensatzes besteht zu 61% aus Nomen. Die meisten dieser Nomen sind zusammengesetzte Wörter, einige davon sind sehr lang, wie zum Beispiel „Reichsdeputationshauptschlussakte“ oder „Hochgeschwindigkeitstransportmittel“. In dem gesamten Vokabular, das aus den Transkriptionen von Whisper stammt, besteht fast die Hälfte (47,8%) aus zehn oder mehr Buchstaben. Der Lemmatisierer von spaCy kann viele dieser Wörter leider nicht auf Stammwörter zurückführen, da sie nicht in seinem Vokabular vorkommen.

In dieser Arbeit wird das Projekt `german_compound_splitter` [77] verwendet, um zusammengesetzte Nomen in ihre Bestandteile zu zerlegen. Dafür werden die Wörter basierend auf dem Free German Dictionary [27] so gut wie möglich in ihre einzelnen Grundwörter zerlegt. Mithilfe dieser Aufspaltung zusammengesetzter Nomen schrumpft das Vokabular um weitere 63% auf nur 73.395 verschiedene Wörter.

4.6 Datenspeicherung

4.6.1 Datenspeicherung SQLite

Für die Analyse werden die MP3-Dateien der einzelnen Podcast-Episoden in einem separaten Ordner gespeichert und die Benennung der originalen Dateien stammt aus der Download-URL. Zur Speicherung der Transkripte wird das relationale Datenbankmanagementsystem

(RDBMS) SQLite [2] eingesetzt. SQLite ist für Datenanalysezwecke sehr gut geeignet, weil es simpel und performant ist, sowie als Open-Source-Projekt kostenlos verwendet werden kann. Im Gegensatz zu anderen RDBMS, wie MySQL oder PostgreSQL, arbeitet SQLite serverunabhängig und speichert alle Tabellen in einer einzigen Datei, was sehr nützlich für den Datenaustausch zwischen verschiedenen Geräten ist. Für einige Operationen, wie die Transkription der Audiodateien oder die Generierung von Embeddings, wird die Hardware eines High-Performance-Clusters der Technischen Hochschule Nürnberg genutzt. Um die Daten zu analysieren und zu nutzen, bietet es sich an, auf Consumer-Hardware zu wechseln, und dafür ist eine gute Portabilität der Datenbank von Vorteil.

SQLite unterstützt Datenmengen bis zu 281 TB [2]. Allerdings wird auf der offiziellen Webseite angegeben, dass ab einer Größe von 1 TB auf serverbasierte RDBMS umgestiegen werden sollte, da die gesamte Datenbank in einem File gespeichert wird und viele Client-Betriebssysteme eine maximale Größe der Dateien vorgeben [2]. Falls das Projekt in der Zukunft auf mehrere Podcastreihen ausgeweitet wird, sollte vor diesem Hintergrund ein Wechsel des DBMS in Betracht gezogen werden.

In der SQLite-Datenbank werden Podcast-Transkripte in zwei Tabellen gespeichert: `transcript_sentences` und `transcript_word_level`.

Die Tabelle `transcript_sentences` enthält die transkribierten Sätze mit folgenden Feldern:

- `filename` (TEXT): Der Name der Audiodatei, aus der das Transkript stammt.
- `sentence` (TEXT): Der vollständige transkribierte Satz.
- `start` (REAL): Die Startzeit des Satzes in der Audiodatei, gemessen in Sekunden.
- `end` (REAL): Die Endzeit des Satzes in der Audiodatei.
- `sentence_id` (INTEGER): Eine eindeutige Identifikationsnummer des Satzes innerhalb des Transkripts.

Die Tabelle `transcript_word_level` dient der detaillierten Erfassung auf Wortebene und umfasst:

- `filename` (TEXT): Analog zu `transcript_sentences`, der Name der Audiodatei.
- `word` (TEXT): Das einzelne transkribierte Wort.
- `start` (REAL): Die Startzeit des Wortes in der Audiodatei.
- `end` (REAL): Die Endzeit des Wortes in der Audiodatei.

Diese Struktur ermöglicht es, die Daten effizient zu speichern. Die gespeicherten Start- und Endzeiten ermöglichen eine Rücküberführung in die entsprechende Stelle im Audio.

4.6.2 Datenspeicherung Vektoren SQLite

Da SQLite nativ keine Listen oder Tabellen als Einträge in einer Datenbank speichern kann, ist es nicht trivial, die Embeddings abzuspeichern [16]. Zunächst wurde die Möglichkeit in Betracht gezogen, jeden einzelnen Eintrag aus einem Embedding-Vektor als separaten Eintrag in einer Tabelle zu speichern.

Das führt allerdings zu sehr ineffizienten Abfragen der Daten und beim Einfügen von Daten limitiert SQLite die Anzahl der Parameter, sodass längere Embeddings umständlich gestückelt abgespeichert werden müssten [36].

Als Nächstes wurde überprüft, ob die Embeddings als serialisiertes Array abgespeichert werden können. Dafür wurde jedes Embedding in einen JSON-String umgewandelt, der dann gespeichert werden soll. Dabei tritt leider das Problem auf, dass die Daten bei der Verwendung wieder deserialisiert werden müssen. Dieser Schritt müsste jedes Mal wiederholt werden, wenn eine Suche stattfindet. Dies dauert sehr lange und ist sehr ineffizient. Für die Deserialisierung von 400.000 Sätzen mit einem 384-dimensionalen Embedding eines Sentence Transformer-Modells auf einem Intel i7 Quad-Core beträgt die Rechenzeit ca. 45 Minuten.

Zur Datenspeicherung der Embedding-Vektoren muss zwischen Dense- und Sparse-Vektoren unterschieden werden, da für beide Embedding-Methoden unterschiedliche Optimierungen in der Speicher- und Retrieval-Funktionalität vorliegen.

4.6.3 Datenspeicherung Dense Vektoren

Um dichte Vektoren abzuspeichern, bietet es sich an, eine separate Vektordatenbank zu nutzen. Eine Vektordatenbank ist eine nichtrelationale Datenbank, die darauf spezialisiert ist, eine effiziente Suche in ungeordneten Informationen zu ermöglichen. Dazu speichert sie die Embedding-Vektoren verschiedener Datenquellen effizient ab und erlaubt Zugriff auf schnelle Suchalgorithmen, wie Approximate Nearest Neighbour-Algorithmen.

Es gibt verschiedene Approximate Nearest Neighbour-Algorithmen, die alle darauf abzielen, den gesamten Suchraum für eine Ähnlichkeitssuche in kleinere Unterräume aufzuteilen. Dabei werden oft Baum-Strukturen verwendet, um die Suche effizienter zu gestalten. Speziell Algorithmen wie Hierarchical Navigable Small Worlds werden von vielen Vektordatenbanken benutzt [51].

In dieser Arbeit wird die Vektordatenbank Chroma [34] verwendet. Chroma ist eine einfache Vektordatenbank, die die Daten sowohl lokal als auch über eine Client-Server-Schnittstelle speichern kann. Das Projekt ist erst im Mai 2022 als Start-up entstanden, verfügt aber mittlerweile über viele Features, die die Datenverwaltung erheblich vereinfachen. Die Datenbank ist dabei mit einer NoSQL-Datenbank zu vergleichen, indem die Daten nicht relational in

Tabellen, sondern in einer Collection als Objekte mit verschiedenen Metadaten gespeichert werden. Jedes dieser Objekte hat eine Document-Eigenschaft, welche den Inhalt des Dokuments repräsentiert. Dieser Inhalt ist in diesem Fall ein Ausschnitt aus einem Transkript, könnte aber auch ein Bild oder eine Audiodatei darstellen. Für jedes Objekt sind zusätzlich der `filename` und die `sentence_id` gespeichert, um diesen Eintrag in der SQLite-Datenbank zu identifizieren.

Außerdem besitzt jedes Objekt einen Embedding-Vektor, der mit dem Dokument assoziiert wird. Innerhalb einer Collection müssen alle Objekte mit derselben Embedding-Methode encodiert werden. Das sichert die Vergleichbarkeit der verschiedenen Vektoren untereinander.

ChromaDB bietet nativen Support für verschiedene Embedding-Modelle von OpenAI, Hugging Face, Cohere, Instructorembedding und JinaAI [12]. Dafür müssen nur das Modell und der plattformspezifische API-Schlüssel angegeben werden und Chroma erstellt automatisch für jedes Dokument das Embedding. Standardmäßig ist das Embedding des Sentence Transformer-Modells all-MiniLM-L6-v2 eingestellt [12]. Alternativ können auch schon vorgefertigte Embeddings eingefügt und die dazugehörige Embedding-Funktion eingetragen werden. In dieser Arbeit wurden die Embeddings von OpenAI und all-MiniLM-L6-v2 schon vorher erstellt und dann erst eingetragen.

Chroma bietet außerdem sehr guten Support für effizientes Retrieval von Dokumenten. Dazu sind schon verschiedene Verfahren der Nearest Neighbour-Suche implementiert.

In diesem Projekt wurde erst in einer späten Phase der Umstieg auf die Vektordatenbank vollzogen, weshalb einige Funktionen, die in dieser Datenbank automatisch integriert sind, noch einmal sehr ausführlich beschrieben werden.

4.6.4 Datenspeicherung Sparse Vektoren

Chroma bietet leider noch keine Unterstützung zur Speicherung von Sparse-Vektoren. Stattdessen wurde das Python-Modul pickle verwendet, welches darauf spezialisiert ist, Python-Objekte effizient als Bytecode zu serialisieren. Es bietet auch die Möglichkeit, Datenstrukturen effizient zu serialisieren bzw. zu deserialisieren. Seit Python 3.8 gibt es auch die Möglichkeit, große NumPy-Arrays effizient zu speichern, was zuvor nur mit der Joblib-Bibliothek möglich war. Pickle speichert die Daten in einem Python-spezifischen Format ab, was die Portabilität auf andere Systeme stark einschränkt. Dieses Format erlaubt es aber, verschiedene Eigenschaften besser zu speichern als JSON (z.B. Pointer-Sharing) [72].

Für den TF-IDF-Algorithmus, bei dem ein Vokabular von mehr als 200.000 Wörtern eine ebensogroße Dimensionalität der Embedding-Vektoren benötigt, würde ein normaler Serialisierungsalgorithmus für die 300.000 Sätze ca. 60 Milliarden Werte abspeichern. Wenn für

jeden dieser Werte eine 32-Bit-Gleitkommazahl als Datentyp abgespeichert würde, wären das ungefähr 240 GB Daten. Ein Großteil dieser Werte (99,9%) sind dabei 0. Ein NumPy-Array kann diese Werte sehr effizient zusammenfassen und mithilfe von pickle kann dieses kompakte Array effizient abgespeichert werden, wodurch eine tatsächliche Speichergröße von ca. 50 MB entsteht. Dies ermöglicht auch ein effizientes Laden und Vergleichen der Embeddings, was für 400.000 Sätze auf einer Intel Quad-Core-i7-CPU nur circa eine Sekunde benötigt.

Kapitel 5

Architektur

5.1 Mikroservicearchitektur

Der zweite Teil des Systems besteht darin, aus der Anfrage der Nutzenden eine passende Podcast-Episode zu erstellen. Die dazu notwendigen Schritte lassen sich zunächst in mehrere kleine Projekte unterteilen. Bei einem so großen Projekt lohnt es sich, eine Mikroservice-Architektur [26] anzustreben, bei der jeder Teil für sich gesehen eine Aufgabe erfüllt und die einzelnen Systeme nur über festgelegte Programmierschnittstellen (APIs) miteinander kommunizieren. Diese Methode wird auch Divide-and-Conquer genannt.

5.2 Anforderungen

Zur Strukturierung des Projektes bietet es sich an, vorab Anforderungen an das System zu stellen. Da das gesamte Projekt als eine Reihe von Mikroservices umgesetzt werden soll, werden folgend für jeden Mikroservice einzelne Anforderungen gestellt.

Zunächst muss die Datengrundlage geschaffen werden. Dafür müssen die Podcast-Episoden heruntergeladen, transkribiert und anschließend alle Sätze in der Datenbank abgespeichert werden. Dieses System muss in der Lage sein, anhand des Titels eines Podcasts bzw. der ID in der Audiothek sämtliche noch nicht in der Datenbank hinterlegten Episoden herunterzuladen, zu transkribieren und abzuspeichern. Dieser Teil wurde in Kapitel 4 erläutert.

Der nächste Mikroservice muss in der Lage sein, diese Transkriptdaten in eine Form umzuwandeln, in der eine Suchfunktion relevante Abschnitte aus diesen Daten extrahieren kann. Dafür soll es die einzelnen Wörter in sinnvolle Abschnitte gruppieren und ein Embedding für jeden Abschnitt berechnen. Die Embeddings müssen dann jeweils in der Vektordatenbank abgespeichert werden.

Für den produktiven Einsatz könnte es vorteilhaft sein, diese beiden Services dynamisch zu gestalten, sodass automatisch neue Podcast-Episoden transkribiert und in der Datenbank gespeichert werden. Für jedes dieser Transkripte könnten dann automatisch Embeddings

berechnet werden. In der Zukunft könnte man darüber nachdenken, diesen Teil komplett auf der Serverseite der ARD Audiothek zu hosten, um diese Daten auch für andere Anwendungen verfügbar zu machen.

Ein weiterer Service soll dann die Suchfunktion übernehmen, indem er Stichworte oder Sätze als Parameter erhält und zu diesen eine bestimmte Anzahl an relevanten Abschnitten zurückgibt. Dafür muss ein Embedding für die Suchbegriffe errechnet werden und dieses Embedding mit allen Embeddings in der Datenbank verglichen werden.

Diese relevanten Abschnitte sollen dann noch weiter von LLMs analysiert werden. Ein weiterer Service erhält die sortierte Liste und nutzt LLMs wie ChatGPT, um zum Beispiel eine Auswahl aus vielen Dokumenten zu treffen, die Reihenfolge der Abschnitte zu bestimmen oder ähnliche Themen herauszufinden.

Der nächste Service muss in der Lage sein, eine Liste an Segmenten entgegenzunehmen und daraus eine Audiodatei zu erzeugen. Dafür muss er die Audiosegmente an den richtigen Stellen aus den originalen Audiodateien schneiden und die Audiosegmente zusammenfügen können.

Als Letztes muss die Audiodatei ausgeliefert werden. Dafür soll zum einen eine API implementiert werden, die eine Anfrage verarbeiten kann und einen Link zu einer Audiodatei zurückgibt. Außerdem soll ein Userinterface in Form einer Webseite aufgebaut werden, welches die Anfrage in einem Formular entgegennimmt und dann die Audioinhalte ausliefert. Die Webseite soll im Browser verfügbar sein und mehrere User-Anfragen verarbeiten können.

5.3 Programmiersprache Python

In dieser Arbeit wird die Programmiersprache Python verwendet. Python zählt zu den am meisten verwendeten Programmiersprachen weltweit und ist laut dem TIOBE-Index im Jahr 2023 sogar die meistverwendete Programmiersprache überhaupt [37]. In dieser Arbeit wird Python verwendet, da es Unterstützung für sehr gute Bibliotheken für Machine Learning und NLP-Anwendungen gibt. Vor allem die Unterstützung für Transformer-Modelle mit der Bibliothek transformers, die NLP-Bibliothek spaCy sowie die Datenverwaltungsbibliotheke pandas sind sehr praktisch. Außerdem bietet die Bibliothek SQLite Unterstützung zur einfachen Anbindung von SQLite-Datenbanken. Dazu ist Python sehr einfach zu verstehen und rechenaufwändige Operationen, wie die Matrixmultiplikationen in der transformers-Bibliothek, sind sehr performant in der Sprache C implementiert.

5.4 Transkript-Segment Ranking

5.4.1 Ranking mit Embeddings

Nachdem bei dem System eine Anfrage der nutzenden Person eingegangen ist, wird diese zunächst mit dem Embedding-Modell in einen Embeddingvektor umgewandelt. Die Wahl des richtigen Embedding-Modells wird in Kapitel 6 beschrieben. Das Ranking der Segmente übernimmt die Vektordatenbank Chroma. Für die Distanzfunktion wird standardmäßig die quadratische L2-Norm verwendet. Nachdem das Ranking erfolgte, werden die besten 10 Ergebnisse zurückgegeben. Dafür wird bei Chroma der Parameter `n_results` gesetzt. Die einzelnen Einträge in Chroma besitzen außerdem einen Parameter `segment_id`, mit dem man die Informationen zu diesem Segment aus der SQLite-Datenbank abrufen kann.

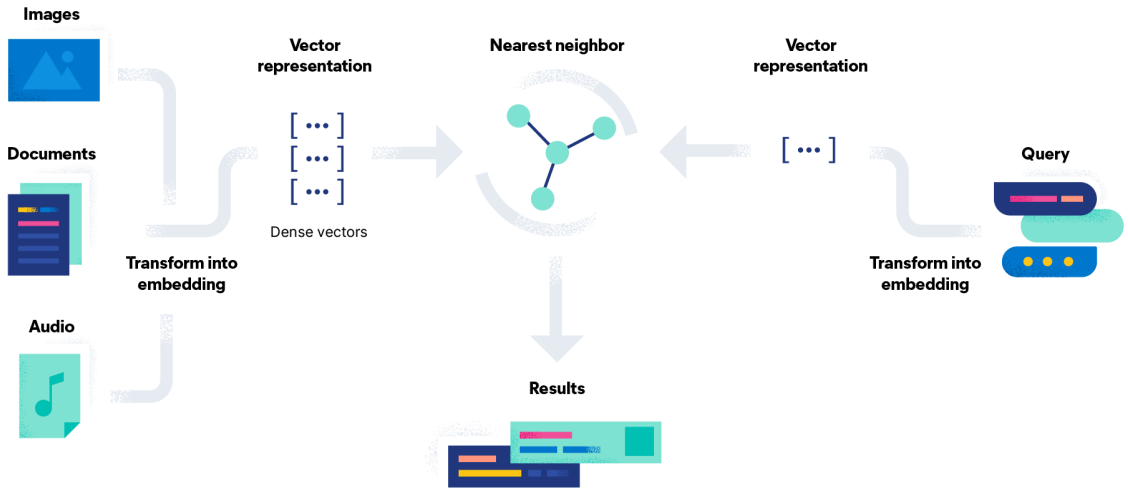


Abbildung 5.1: Retrieval der relevanten Segmente [44]

5.4.2 Anreicherung der Segmente

Das Ranking der einzelnen Sätze liefert eine Auswahl der besten Kandidaten für den Podcast. Einzelne Sätze, die aus verschiedenen Podcast-Episoden stammen, ohne Kontext hintereinander abzuspielen, bietet für den Zuhörenden nur ein geringes Hörerlebnis. Es fehlt der Kontext zu den einzelnen Informationen. Zum Beispiel liefert die Suchanfrage „Geschichte Amsterdam“ mit dem TF-IDF-Ansatz Sätze wie „Amsterdam, das bedeutet unbeschwerte Kinderjahre“ oder „Amsterdam gefiel ihm“. Die Sätze an sich bieten kaum interessante Informationen. Den Zuhörenden stellen sich sogar noch mehr Fragen, zum Beispiel, für wen Amsterdam unbeschwerte Kinderjahre bedeutete oder wem Amsterdam gefiel.

Um den Zuhörenden mehr Informationen zu ermöglichen, kann die Größe der einzelnen Segmente angepasst werden. Dazu werden mehrere Ansätze betrachtet.

Der einfachste Ansatz ist, für jeden einzelnen Satz die umgebenden Sätze davor und dahinter miteinzubeziehen. Die Anzahl der umgebenden Sätze ist dann ein Hyperparameter dieses Systems und kann vom Nutzer durch den Parameter Segmentlänge eingestellt werden, welcher die Anzahl der Sätze in einem Segment angibt. Falls der auszuschneidende Satz am Anfang oder am Ende des Transkripts vorkommt, und die Segmentlänge so eingestellt ist, dass mehr Sätze als möglich dem Segment hinzugefügt werden sollten, so wird das Segment an der Transkriptgrenze abgeschnitten. Die Evaluation der besten Segmentlänge wurde in dieser Arbeit nicht ausgeführt, als Defaultwert ist für die Segmentlänge 5 eingestellt.

Ein weiterer Ansatz wäre auch hier ein mächtiges LLM, wie ChatGPT, zu benutzen, um die richtige Segmentlänge für jedes Segment individuell einzustellen. Dazu wird das LLM instruiert, aus dem gesamten Transkript einer Podcast-Episode und der dazugehörigen Anfrage die Anzahl der Kontextsätze auf das Wesentliche zu beschränken. Ein Beispiel-Prompt ist zum Beispiel hier (8) nachzulesen. Dieser Ansatz führt in der Regel zu längeren Abschnitten, was es schwerer macht, die Zeitvorgaben zu erfüllen.

5.4.3 Reranking mit ChatGPT

Nachdem nun die einzelnen Sätze zu größeren Segmenten erweitert wurden, kann auch die Reihenfolge der einzelnen Segmente angepasst werden. Wenn eine Person sich beispielsweise über die „Geschichte von Amsterdam“ informieren will, kommen dazu verschiedene Ausschnitte aus der Zeit der Gründung der Niederlande im Jahr 1581, den Besuchen Peter des Großen in Amsterdam im Jahr 1697 oder der Verfolgung der Juden im Zweiten Weltkrieg. Die Reihenfolge dieser Ereignisse wird im Retrieval-Schritt dadurch bestimmt, wie stark die Ähnlichkeit zwischen diesen Segmenten und der Frage ist. In diesem Fall wäre es sehr nützlich, die Reihenfolge so umzuändern, dass die Segmente zeitlich sortiert wären. In anderen Fällen könnte es sinnvoll sein, dass Segmente hintereinander erscheinen, die einen inhaltlichen Zusammenhang besitzen. Zum Beispiel würde zum Thema „Mauerfall Berlin“ mehrere Segmente vorkommen, die sich auch mit der DDR befassen. In diesem Fall wäre es für Zuhörende spannend, diese Segmente hintereinander zu platzieren, um den Podcast flüssiger und zusammenhängender zu gestalten.

Ein solches Reranking der einzelnen Segmente kann mithilfe von leistungsstarken LLMs wie ChatGPT erreicht werden. Die einzelnen Segmente werden nummeriert an die Inferenz-API von ChatGPT geschickt und das LLM soll die neue Reihenfolge als Liste zurückgeben. Dafür wird der JSON-Modus von ChatGPT genutzt, welcher das LLM dazu bringt, die Antwort als JSON-String zurückzugeben. Ein Beispiel-Prompt kann im Anhang (8) nachgelesen werden.

Da dies eine relativ einfache Aufgabe ist, in der nur eine Liste mit wenigen Zahlen zurückgegeben werden muss, wird in diesem Schritt auf Techniken des Prompt-Engineerings verzichtet.

5.5 Audio-Zusammensetzung

Für die Bearbeitung von Audiodateien in Python bietet sich das Python-Modul Pydub [38] an. Mit diesem Modul kann ein Audiofile ähnlich wie ein Array behandelt werden. Wenn sich zum Beispiel in einem Audiofile ein wichtiges Segment von Sekunde 34 bis Sekunde 64 erstreckt, kann dort einfach die Start- und Endzeit in Array-Schreibweise angegeben werden. Im Hintergrund verwendet dieses Modul Software-Bibliotheken des Softwareprojekts FFmpeg [24], welche umfangreichen Support für die Bearbeitung fast aller Audioformate besitzt. Für die Zeitstempel der Start- und Endzeit jedes Audiosegments nehmen wir die Daten aus den vorher aus der SQLite extrahierten relevanten Segmenten. Diese werden dann als extra Audiofiles als WAV-Dateien in einem separaten Ordner abgespeichert, um einem Qualitätsverlust durch die Codierung des verlustbehafteten Codierungsformats MP3 vorzubeugen.

Um die Audios nun wieder zusammenzusetzen, verwenden wir das gleiche Modul Pydub. Es bieten sich mehrere Möglichkeiten an, die Audiosegmente zusammenzusetzen. Der primitivste Ansatz wäre, die Segmente einfach ohne Pause hintereinander abzuspielen. Die einzelnen Segmente beginnen und enden meist abrupt und das Hintereinanderschalten mehrerer Segmente ohne Pause führt zu Verwirrungen, wo ein Segment endet und wo das nächste anfängt. Dafür bietet sich ein kurzer Signalton zwischen den einzelnen Audiosegmenten an. Dieser sollte nicht nervig sein, da er dem Zuhörenden öfter vorgespielt wird. In dieser Arbeit wurde ein kleiner Jingle, der zum Anfang einer Episode abgespielt wird, als Signalton verwendet.

Eine weitere Möglichkeit wäre, zwischen jedem Segment der zuhörenden Person eine kurze Vorstellung der Episode und der Sprecherin bzw. des Sprechers zu ermöglichen oder sogar Kontext zu dieser zu geben. Dazu könnte eine kurze Einleitung zu jedem Segment mit einer synthetisch generierten Stimme erstellt werden. Dieser Ansatz wurde hier aber nicht weiter verfolgt.

5.6 Weiterführende Themenvorschläge

Um den Zuhörenden die Möglichkeit zu geben, auf verschiedene Themen, die im Podcast erwähnt wurden, weiter einzugehen, wurden entsprechende Themenvorschläge mithilfe von ChatGPT generiert. Wenn zum Beispiel in einem Podcast zur „Geschichte von Amsterdam“ einzelne Segmente über die Anfangszeit der Niederlande handeln, könnte man sich

in einer weiterführenden Episode zum Beispiel über die „Gründung der Niederlande“ informieren. Zu diesem Zweck werden für jede generierte Podcast-Episode fünf weiterführende Themenvorschläge von ChatGPT generiert, die zu den Segmenten passen. Dazu wird wieder ChatGPT-3.5 im JSON-Modus verwendet. Ein Beispielprompt ist hier (8) zu lesen. Die Themen werden explizit auf ein bis drei Wörter beschränkt, da sonst manchmal ganze Sätze als Themenvorschläge geführt werden, wie zum Beispiel „Die Erfahrungen von Juden in Amsterdam während der Nazibesetzung“. Die Antwort von ChatGPT besteht aus einem JSON-Array, welches die weiterführenden Themen enthält. Die einzelnen Themen können je nach Auslieferungsart des Systems zum Beispiel als Buttons angezeigt werden, die eine erneute Suche auslösen, sobald eine nutzende Person ihn drückt.

5.7 Auslieferung

Das System wird in diesem Prototyp als Webservice zur Nutzung angeboten. Für das Deployment dieses Tools wurde das Webframework Flask genutzt. Flask ist ein Web Service Gateway Interface-Server (WSGI), welcher das Bereitstellen von Webseiten über die WSGI-Schnittstelle ermöglicht, die dann von einem HTTP-Server ausgeliefert werden können. In dieser Arbeit wird dafür der WSGI-HTTP-Server gunicorn [30] verwendet. Gunicorn ist für Unix-Systeme geeignet, einfach zu konfigurieren und bietet Support für mehrere Worker, die mehrere Anfragen gleichzeitig bearbeiten können.

Für die Auslieferung wurde eine API und eine Webseite entwickelt.

5.7.1 API

Über die API kann der Podcast-Generator einfach von anderen Anwendungen aufgerufen werden. Die API wird aufgerufen über eine GET-Request auf die Unterseite /api.

Als Parameter akzeptiert die API einen String „text“, welcher die Frage des Nutzers angibt. Außerdem kann über den Parameter „time“ ein Integerwert angegeben werden, welcher die Zeit des Podcasts in Minuten angibt. Der optionale Parameter „segment-length“ kann als Integer übergeben werden und bestimmt die Länge der einzelnen Segmente.

Die API gibt bei erfolgreicher Verarbeitung den Statuscode 200 zurück und liefert als Inhalt einen JSON-String, der eine einzige Variable „url“ enthält. Diese URL verweist auf das generierte MP3-File, das ab diesem Zeitpunkt unter der Adresse /static/audios verfügbar ist.

Als Demonstration wurde außerdem ein Telegram-Bot entwickelt, der die API benutzt, um damit die automatische Podcast-Episoden zu erstellen. Der Telegram-Bot ist öffentlich verfügbar und kann über die Adresse `https://t.me/PodcastGenerator` gefunden werden.

5.7.2 Design der Webseite

Die Webseite des Podcast-Generators bietet einen einfachen Weg, das System zu benutzen. Es gibt die beiden Inputfelder Thema und Zeit, und einen Startbutton, bei dem die Generierung der Podcasts startet. Nachdem die Podcast-Episode fertig generiert wurde, wird der vorher ausgegraute Audioplayer aktiv. Dann können Nutzende die Audiodatei mithilfe des Players anhören. Außerdem befindet sich neben dem Audioplayer eine Box mit den Transkriptabschnitten des Podcasts, sortiert nach der Erscheinung in der generierten Episode. Diese Transkriptabschnitte können zur Navigation innerhalb der Podcast-Episode verwendet werden.

Das Design der Webseite beruht auf einem Template von [22]. Dieses Template wurde erweitert durch einen Suchbereich und einen Transkriptbereich. Insgesamt soll es den nutzenden Personen eine intuitive Anwendung des Tools erlauben und die Möglichkeit liefern, die Parameter Thema, Zeit und Segmentlänge der Episode festzulegen.

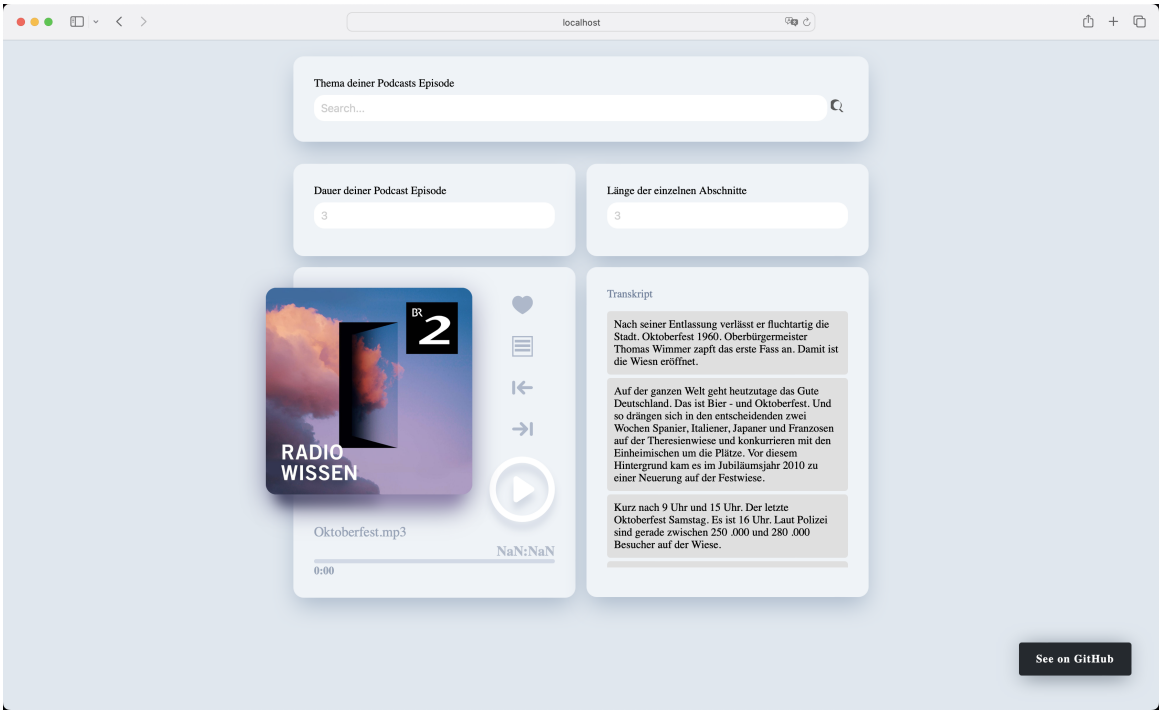


Abbildung 5.2: Screenshot der Webseite

Kapitel 6

Evaluation verschiedener semantischer Verfahren

6.1 Ausgangslage

In diesem Kapitel werden verschiedene Embedding-Modelle miteinander verglichen, um das Modell mit den besten Retrieval-Eigenschaften zu ermitteln. Wie in Kapitel 3 beschrieben, gibt es bereits Rankings von verschiedenen Embedding-Modellen hinsichtlich ihrer Retrieval-Funktionalität. Leider gibt es noch nicht genügend deutsche Datensätze, um die Retrieval-Eigenschaften der Modelle für Texte in deutscher Sprache zu bewerten. Spezielle Datensätze sind zum Zeitpunkt dieser Arbeit nur für Englisch, Chinesisch, Französisch und Polnisch verfügbar [56].

Die explizite Bewertung im Kontext der eigenen Daten hat außerdem den Vorteil, dass sie die Eigenschaften der Modelle für diese Daten noch genauer bewerten kann.

6.2 Bewertungsmethoden

Für die Bewertung eines Information Retrieval Systems können verschiedene Methoden verwendet werden.

Die Güte eines Information Retrieval Systems kann mithilfe des normalized discounted cumulative gain (NDCG) [20] beschrieben werden. Dafür muss allerdings für jedes Thema bereits eine vorgefertigte Vergleichsliste, die Ground Truth, vorhanden sein, mit der man dann die Ergebnisse vergleichen kann. Diese Vergleichsliste, oder Ground Truth, wird normalerweise manuell erstellt. Für 400.000 Segmente über 9 verschiedene Themen ein manuelles Ranking durchzuführen, ist leider im Rahmen dieser Arbeit nicht durchführbar.

In dieser Arbeit wird der Ansatz verfolgt, das Ranking der einzelnen Segmente mithilfe von LLMs zu erstellen. Leistungsstarke LLMs, wie GPT-4, werden mittlerweile in der Forschung für automatische Bewertungsaufgaben von Texten eingesetzt [59] [62].

6.3 Auswahl der Embedding-Modelle

Für die Bewertung werden vier verschiedene Embedding-Modelle miteinander verglichen. Die berücksichtigten Modelle decken sowohl Sparse- als auch Dense-Embedding-Ansätze ab, was eine breite Untersuchungspalette bietet. Leider konnten viele open-source Modelle nicht evaluiert werden, da die Hardwareanforderungen zu hoch waren, bzw. durch schlechte Hardware der Embeddingprozess zu lange gedauert hätte. Zum Beispiel würde das Embedding von 400.00 Sätzen mit dem besten Modell SFR-Embedding-Mistral [79] auf dem MTEB auf einem MacBook Pro 2016 mit einem Intel Core i7 über 300 Stunden dauern. Der TF-IDF-Algorithmus wurde ausgewählt, da er ein klassischer Vertreter für Sparse-Embeddings ist und als Baseline herangezogen wird. Er hat eine grundlegend andere Retrievaleigenschaften als die Dense-Vektoren, indem er nach lexikalischen Eigenschaften sucht anstatt nach semantischen Ähnlichkeiten.

Als weiteres Modell wurde das Sentence Transformer Modell all-mini-LM-L6-v2-Modell [54] ausgewählt. Es ist aufgrund seiner geringe Größe und der relativ guten Leistung das am meisten heruntergeladene Modell in der Kategorie Sentence-similarity auf der Plattform Huggingface [55].

Das Modell voyage-lite-02-instruct-Modell [21] belegt den zweiten Platz der Massive Text Embedding Benchmark, sowohl in der Leistung über alle einzelnen Benchmarks, als auch in der Kategorie Retrieval. Die Embeddings können dabei über eine API-Schnittstelle erhalten werden. Das bestplazierte Modell, konnte leider nicht benutzt werden, da es zu groß ist, um auf der verfügbaren Hardware laufen zu können.

Ein weiteres Embedding-Modell, bei dem die Embeddings per API übergeben werden, ist das text-embeddings-small-Modell [64] von OpenAI. Dieses Modell erzielt nur Platz 27 in der Kategorie Retrieval auf dem MTEB [56], wird aber als Vergleich herangezogen.

6.4 Vorgehensweise

Die Bewertung des IR-Systems erfolgt auf einer Auswahl verschiedener Themen. Das System berechnet für jedes Thema ein Embedding, vergleicht es mit den Embeddings in der Datenbank und erstellt daraus eine Rangliste. Für jedes Thema wird dann eine sortierte Liste der Segmente zurückgegeben, welche zur Evaluierung des Modells benutzt wird.

Die Aufgaben, nach denen die Embeddings evaluiert werden sollen, sind in zwei Kategorien eingeteilt: „topical“ und „re-findings“ [40]. Bei der Kategorie „topical“ wird verlangt, passende Segmente zu einem bestimmten Thema zu finden. Dazu werden 30 verschiedene Themen von ChatGPT generiert (siehe Abschnitt 8), die aus den Bereichen Geschichte, Naturwissenschaft, Gesellschaft oder Philosophie stammen. Als Vergleich werden zudem 30

verschiedene Fragen generiert (siehe Abschnitt 8), um zu untersuchen, ob eine größere Spezifizierung des Themas die Retrieval-Eigenschaften verbessern kann. Für jedes Modell werden die 10 Segmente mit der größten Ähnlichkeit ausgewählt. Zur Bewertung wird das Modell ChatGPT-4 verwendet. Dafür werden die Segmente zusammen mit der originalen Anfrage in einem Prompt an das Modell geschickt. ChatGPT-4 bewertet daraufhin die Segmente nach den drei Metriken Precision, Cohesion und Uniqueness. Bei der Precision wird die inhaltliche Relevanz der Segmente zu dem Thema auf einer Skala von null bis fünf bewertet. Die Cohesion bewertet die inhaltliche Zusammengehörigkeit der einzelnen Segmente zueinander auf einer Skala von null bis drei. Das dritte Bewertungskriterium ist die Uniqueness, bei der untersucht wird, ob sich die einzelnen Segmente inhaltlich überlappen oder jedes einen eigenen Inhalt darstellt. Diese Metrik wird auf einer Skala von null bis zwei evaluiert.

Beim „re-finding“ geht es darum, einen zuvor bekannten Audioinhalt wiederzufinden. Dabei werden als Anfrage Suchworte gestellt, die den Titel einer speziellen Episode beschreiben. Für jedes Embedding-Modell werden die drei Segmente mit dem größten Ähnlichkeits-Score ermittelt. Es wird untersucht, ob unter den Abschnitten auch Segmente sind, die aus der gesuchten Episode stammen. Wenn der erste Treffer aus dieser Episode stammt, werden drei Punkte vergeben, wenn der zweite Treffer übereinstimmt, zwei Punkte und falls nur der letzte Treffer die richtige Episode markiert, wird ein Punkt vergeben. Stammt keines der Segmente aus diesem Podcast, wird kein Punkt vergeben. Dabei werden wieder 30 verschiedene Themen evaluiert.

6.5 Auswertung

6.5.1 Auswertung topical

Abbildung 6.1 veranschaulicht die Bewertungen von Segmenten bezogen auf unterschiedliche Themen. Abbildung 6.2 stellt die Bewertungen von Segmenten zu konkreten Fragestellungen dar.

In Abbildung 1 erreicht das TF-IDF-Embedding die höchste Precision, dicht gefolgt vom text-embeddings-small von OpenAI und dem all-mini-LM-L6-v2-Modell von OpenAI. Das voyage-lite-02-instruct von Voyage AI zeigt hingegen die niedrigste Precision. In Abbildung 2 verzeichnet das text-embeddings-small von OpenAI die beste Precision, wohingegen das all-mini-LM-L6-v2-Modell einen merklichen Abfall in der Präzision im Vergleich zu Abbildung 1 aufweist.

Es lässt sich erkennen, dass das TF-IDF-Embedding insgesamt gute Ergebnisse bei der Themensuche erzielt, besonders hinsichtlich der Precision und Uniqueness. Das deutet darauf

hin, dass TF-IDF besonders effektiv darin ist, relevante und einzigartige Inhalte zu extrahieren. Das text-embeddings-small von OpenAI demonstriert eine durchgängig solide Leistung in beiden Abbildungen und könnte als robustes Modell für verschiedene Suchanfragen angesehen werden. Die Modelle voyage-lite-02-instruct von Voyage AI und all-mini-LM-L6-v2-Modell zeigen schwächere Ergebnisse, vor allem bei der Beantwortung spezifischer Fragen.

Bezüglich der Precision stellt sich heraus, dass voyage-lite-02-instruct von Voyage AI geringere Werte als das text-embeddings-small von OpenAI erreicht. Das TF-IDF-Embedding zeigt bei Themen, die mit 1-3 Wörtern beschrieben werden, eine höhere Präzision als bei Fragen. Beim all-mini-LM-L6-v2-Modell wird eine Verminderung der Precision festgestellt, wenn es um die Beantwortung von Fragen im Vergleich zu Themen geht.

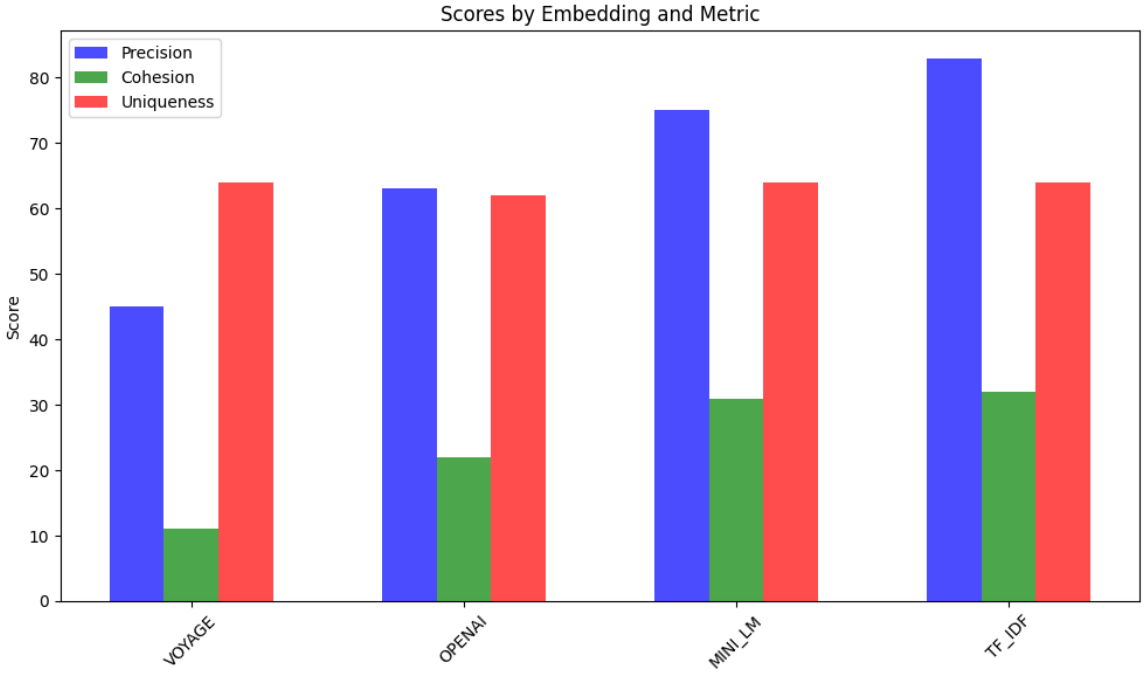


Abbildung 6.1: Vergleich der verschiedenen Embedding-Modelle (Themen 1-3 Wörter)

6.5.2 Auswertung re-finding

In Abbildung 6.5 wird die Leistungsfähigkeit unterschiedlicher Embedding-Modelle bei der Aufgabe des Wiederfindens von spezifischen Audioinhalten dargestellt. Die Auswertung basiert auf einem Scoring-System, bei dem Suchworte zur Identifikation von Episodentiteln genutzt werden. Die Modelle werden danach bewertet, wie gut sie in der Lage sind, die richtige Episode innerhalb der drei höchsten Similarity Scores zu identifizieren.

Das text-embeddings-small von OpenAI erweist sich als das leistungsfähigste Modell und erreicht die höchsten Werte. Dies deutet auf eine hohe Effizienz des Modells beim Zuord-

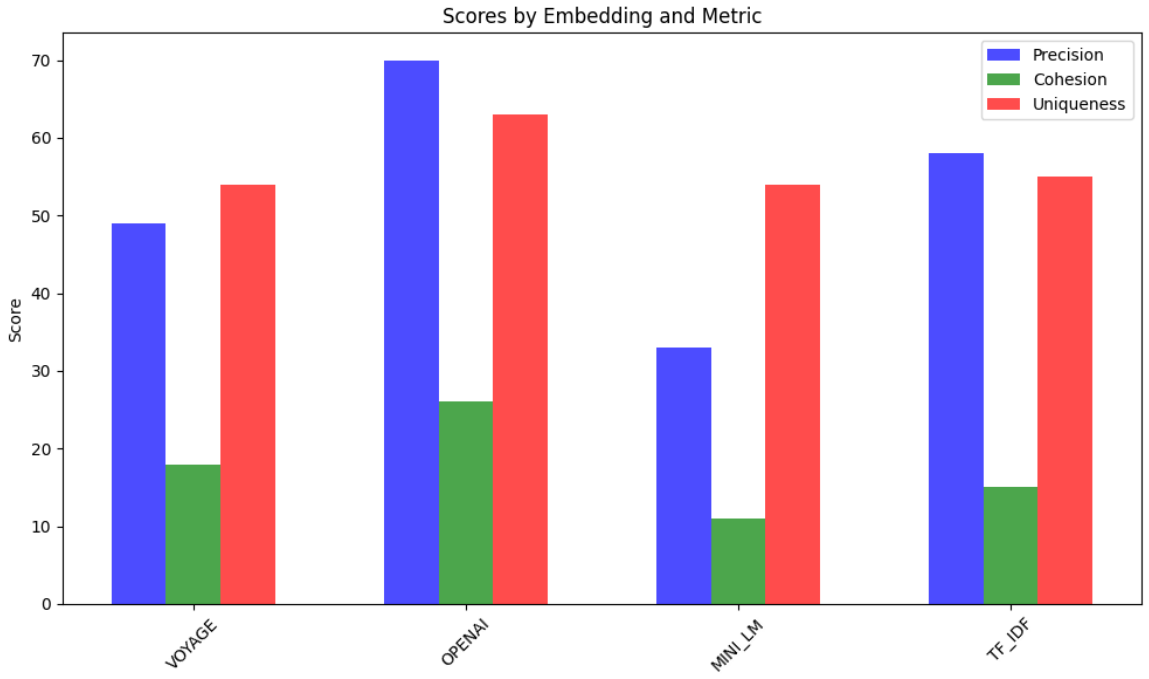


Abbildung 6.2: Vergleich der verschiedenen Embedding-Modelle (Fragen)

nen von Suchanfragen zu den entsprechenden Audiosegmenten hin. Das all-mini-LM-L6-v2-Modell und das TF-IDF-Embedding zeigen ähnliche Ergebnisse und liegen im mittleren Bereich des Rankings. Das voyage-lite-02-instruct von Voyage AI hingegen zeigt deutliche Schwächen und erreicht die niedrigsten Punktzahlen in dieser spezifischen Aufgabe des re-finding.

Es ist bemerkenswert, dass das TF-IDF-Modell, welches typischerweise auf Keyword-Suche spezialisiert ist, nicht die Spitzenposition einnimmt. Dies könnte auf die Art der Suchanfragen, die Komplexität der Daten oder auf die Spezifität der Themen zurückzuführen sein, die möglicherweise eine tiefere semantische Analyse erfordern, als sie durch reinen Keyword-Abgleich erreicht werden kann.

6.6 Diskussion

Die in Abbildung 6.5 präsentierten Ergebnisse zeigen einige überraschende Trends in Bezug auf die Leistungsfähigkeit der untersuchten Embedding-Modelle. Die unterdurchschnittliche Performance des voyage-lite-02-instruct Modells von Voyage AI ist dabei besonders bemerkenswert. Eine mögliche Erklärung für dieses Phänomen könnte sein, dass voyage-lite-02-instruct für die Verarbeitung größerer Segmente optimiert ist, da bei der Embeddinggenerierung zunächst ein weiterer Prompt an die Sätze angehangen wird [21]. Dies könnte darauf hindeuten, dass die voyage-lite-02-instruct Architektur weniger geeignet für Aufgaben ist,

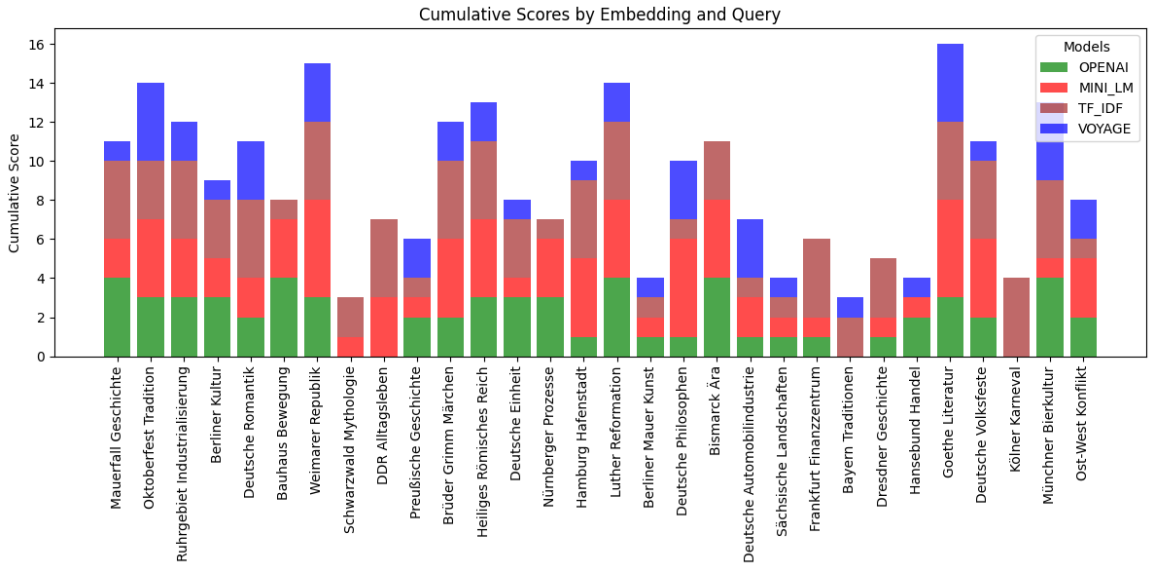


Abbildung 6.3: Vergleich der Fragen (Themen 1-3 Wörter)

die eine präzise und enge Abstimmung zwischen Suchanfragen und spezifischen Segmenten erfordern, wie es bei der re-finding Aufgabe der Fall ist.

Das text-embeddings-small Modell von OpenAI erweist sich als das effektivste über die verschiedenen evaluierten Szenarien hinweg. Die Embeddings von OpenAI sind anscheinend eine ausgewogene und anpassungsfähige Lösung für eine Vielzahl von Suchaufgaben. Es scheint, dass die allgemeine Architektur und die Trainingsmethodik, die hinter den Embeddings von OpenAI stehen, für die unterschiedlichen Herausforderungen der Inhaltsrecherche besonders gut geeignet sind.

Die Ergebnisse könnten weitere Untersuchungen motivieren, ob Dense Embeddings für Themen durch eine Augmentierung der Nutzeranfragen mit Unterstützung durch ChatGPT verbessert werden könnten. Dazu gibt es bereits aktive Forschung [80][15]. Die Erweiterung von Suchanfragen durch ChatGPT könnte semantischen Embeddings mehr Informationen geben, wodurch sie eventuell bessere Ergebnisse liefern könnten.

6.7 Schlussfolgerung

Die Untersuchung der Embeddings hat gezeigt, dass die Auswahl und Konfiguration von Embedding-Modellen entscheidend für die Effizienz und Präzision von Information-Retrieval-Systemen ist. Während das text-embeddings-small von OpenAI eine robuste Leistung über verschiedene Aufgaben hinweg bietet, zeigen sich bei den anderen Embedding-Methoden Unterschiede.

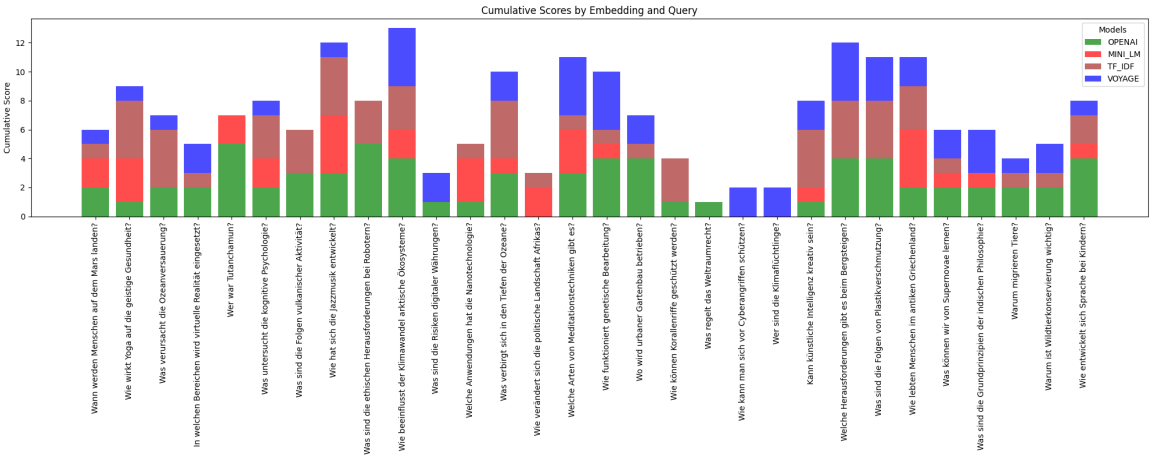


Abbildung 6.4: Vergleich der Fragen (Fragen)

Das TF-IDF-Embedding und das all-mini-LM-L6-v2-Modell zeigten eine mittelmäßige Performance, was Raum für Verbesserungen lässt. Besonders auffällig war die geringe Leistung des voyage-lite-02-instruct von Voyage AI, was auf eine mögliche Inkompatibilität mit der Anforderung des genauen Matchings bei re-finding Aufgaben hindeutet.

Aus diesen Ergebnissen lässt sich schließen, dass für die Optimierung von Suchalgorithmen eine tiefergehende Betrachtung der verschiedenen Embedding-Modelle erforderlich ist.

6.7.1 Kritische Reflexion

Die Untersuchung unterstreicht die Bedeutung der Segmentgröße in der Datenbank für die Leistung der Embedding-Modelle. Größere Segmente könnten den Kontext für Suchanfragen verbessern, was vor allem bei komplexen Anfragen hilfreich wäre. Zukünftige Experimente sollten sich daher mit der optimalen Balance von Segmentgröße und Suchpräzision befassen.

Eine Vorverarbeitung der Suchanfragen, möglicherweise durch ein Modell wie ChatGPT, könnte ebenso vorteilhaft sein, um Anfragen besser auf die Daten abzustimmen und so die Suchergebnisse zu verfeinern. Auch das Prompt Engineering bietet Raum für Verbesserungen; die Nutzung von Few-shot Prompts [7] könnte helfen, das Verständnis des Modells für spezifische Aufgaben zu schärfen und so die Bewertungsgenauigkeit zu steigern.

Beim re-finding könnten Methoden entwickelt werden, die spezifische Abschnitte innerhalb einer Datei identifizieren. Dies würde zu einer erhöhten Genauigkeit führen, da nicht nur die richtige Episode, sondern der exakte Abschnitt gefunden wird.

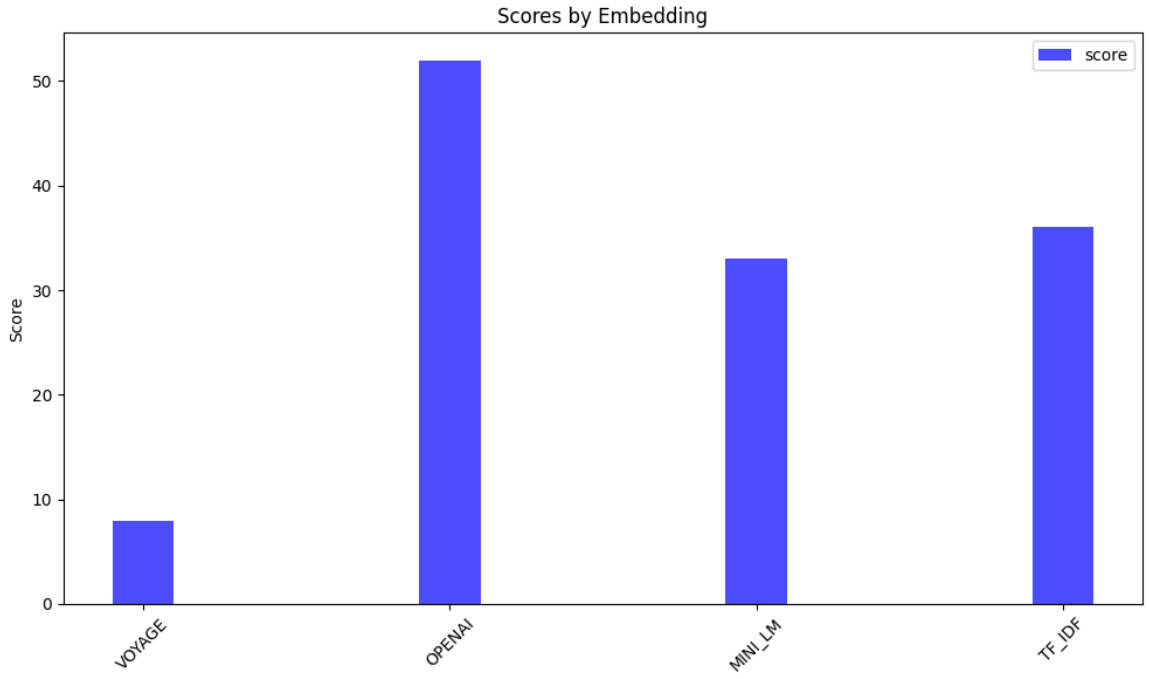


Abbildung 6.5: Vergleich der Embedding-Modelle (re-finding)

Diese Arbeit liefert wichtige Einsichten, zeigt jedoch auch, dass die fortlaufende Optimierung der Suchalgorithmen notwendig ist, um die Treffsicherheit und Effektivität von Suchsystemen zu verbessern. Zukünftige Forschung sollte diese Aspekte weiter untersuchen.

Kapitel 7

Ausblick

Dieses Kapitel bietet einen Ausblick auf mögliche Verbesserungen des vorgestellten Podcast-Generierungssystems und skizziert zukünftige Entwicklungspfade.

Für eine weitergehende Optimierung des Systems könnten zunächst die zugrunde liegenden Datenbestände erweitert werden. Mehr Daten würden die Vielfalt und Tiefe des Inhalts erhöhen, was zu einer qualitativ besseren und nuancierteren Podcast-Produktion führen kann. Des Weiteren wäre es sinnvoll, die Segmentierung der Daten zu überarbeiten. Größere und überlappende Segmente könnten dabei helfen, den Kontext besser zu bewahren und die Verknüpfung von relevanten Inhalten zu erleichtern.

Ein weiteres wichtiges Feld für zukünftige Entwicklungen ist die Exploration neuer Embedding-Modelle. Durch das Testen und Integrieren verschiedener innovativer Embeddingmethoden könnte die Effizienz und Präzision des Retrievalprozesses deutlich verbessert werden.

Die Zukunft des Information Retrievals könnte sich jedoch grundlegend ändern, wie die Forschung im Bereich "Mistral of Experts"[39] nahelegt. Hier wird darauf hingewiesen, dass durch den Einsatz innovativer Technologien, die große Kontextmengen verarbeiten können, die traditionellen Methoden des Information Retrievals möglicherweise bald überholt sein könnten. Insbesondere die neuen Modelle wie Gemini1.5 Pro [65], die über die Kapazität verfügen, mehr als 10 Millionen Token zu verarbeiten, zeichnen sich durch eine außergewöhnlich hohe Retrieval-Rate aus. Diese Entwicklungen deuten darauf hin, dass wir an der Schwelle zu einer neuen Ära im Bereich des Information Retrievals stehen, in der die Grenzen der aktuellen Systeme signifikant erweitert werden könnten.

Kapitel 8

Zusammenfassung

In dieser Bachelorarbeit wurde die Entwicklung eines Systems zur Generierung von Podcasts detailliert beschrieben. Der Prozess umfasste mehrere Kernschritte: von der Datenbeschaffung und -verarbeitung über die Bildung von Segmenten, die Implementierung von Retrievalfunktionen bis hin zur Audioproduktion und der finalen Auslieferung des Podcasts. Ein wesentlicher Schwerpunkt lag auf der Nutzung von Large Language Models (LLMs), deren Potenzial in verschiedenen Phasen des Entwicklungsprozesses ausgeschöpft wurde. Darüber hinaus erfolgte eine gründliche Evaluation verschiedener Embedding-Methoden, um deren Effektivität und Eignung für das vorgestellte Podcast-Generierungssystem zu bestimmen.

Das Quellcode des Projektes ist hier [25] öffentlich verfügbar.

Abbildungsverzeichnis

3.1 Transformer Architektur aus [87] 19
3.2 Die einzelnen Datenbestände in MTEB aus [58] 20
3.3 Screenshot des MTEB Leaderboards [56] 20

4.1 Länge der MP3 Dateien 24
4.2 Anzahl der Wörter pro Transkript 27

5.1 Retrieval der relevanten Segmente [44] 37
5.2 Screenshot der Webseite 41

6.1 Vergleich der verschiedenen Embedding-Modelle (Themen 1-3 Wörter) 45
6.2 Vergleich der verschiedenen Embedding-Modelle (Fragen) 46
6.3 Vergleich der Fragen (Themen 1-3 Wörter) 47
6.4 Vergleich der Fragen (Fragen) 48
6.5 Vergleich der Embedding-Modelle (re-finding) 49

Anhang

GraphQL Download-Links

GraphQL Abfrage um alle Download-Links zu den Episoden des Podcasts „radioWissen“ zu erhalten. Die Programset-ID verweist auf den speziellen Podcast.

```
{
    programSet(id: 5945518) {
        title
        items(
            orderBy: PUBLISH_DATE_DESC
            filter: {
            isPublished: {
                equalTo: true
            }
            }
        ) {

            nodes {
                title,
                audios {
                    downloadUrl
                }
            }
        }
    }
}
```

GraphQL Metadaten

GraphQL Abfrage um alle Metadaten zu den Episoden des Podcasts „radioWissen“ zu erhalten. Die Programset-ID verweist auf den speziellen Podcast.

```
{
    programSet(id: 5945518) {
    title
    items(
        orderBy: PUBLISH_DATE_DESC
        filter: {
        isPublished: {
            equalTo: true
        }
        }
    ) {

        nodes {
          title,
          keywords
          publishDate
          description
        }
    }
    }
}
```

GraphQL Transkripte

GraphQL Abfrage um alle Transkripte zu den Episoden des Podcasts „radioWissen“ zu erhalten. Die Programset-ID verweist auf den speziellen Podcast.

```
{
programSet(id: 5945518) {
    title
    items(
    orderBy: PUBLISH_DATE_DESC
    filter: {
        isPublished: {
        equalTo: true
        }
    }
```

```
    ) {
    nodes {
        transcript {
        data
        }
        audios {
        downloadUrl
        }
    }
    }
}
}
```

ChatGPT Segmente ranken

Ein Beispielprompt um Segmente von ChatGPT sortieren zu lassen. Die einzelnen Segmente werden automatisch ergänzt.

Systemprompt: *Bringe die Abschnitte in eine logische Reihenfolge. Antworte in JSON Format als ein Array 'Reihenfolge' in welchem nur die Nummern der Segmente stehen.*

Userprompt:

1: Und das waren ja hochgebildete und bedeutsame Personen unter ihnen, die da für das holländische Leben eine große Rolle spielten. Zentrum der neu gegründeten Republik war Amsterdam. Das mächtige, prächtige Amsterdam. Die reichste Stadt der Welt. Metropole eines weltumspannenden Handelsimperiums und so multikulturell wie vor ihm nur das alte Rom gewesen war. Alle Pracht und alle Luxusgüter aus aller Herren Länder kamen nach Amsterdam.

2: Keine falsche Assoziation. Peter der Große hat in seiner Jugend die Niederlande besucht. Amsterdam gefiel ihm. Petersburg zu bauen, war aber keine nostalgische Entscheidung, sondern hatte machtstrategische Gründe. Dem Zaren lag Moskau als Hauptstadt zu weit im Osten. Der Nordriese, wie Peter manchmal genannt wird, wollte sein Reich europäischer machen, auch durch den Bau einer neuen Hauptstadt.

3: Bis zu meinem vierten Lebensjahr wohnte ich in Frankfurt. Da wir Juden sind, ging dann mein Vater 1933 in die Niederlande. Amsterdam, das bedeutet unbeschwerte Kinderjahre. Bis die Nazis im Mai 1940 auch die Niederlande besetzen. Die Schlinge zieht sich zu für die Juden dort. Eine antisemitische Verordnung jagt die andere.

Beispielantwort: (Die Antworten können variieren, da für die Inferenz kein Seed benutzt wurde)

```
{
    "Reihenfolge": [2, 1, 3]
}
```

ChatGPT Themen extrahieren

Ein Beispielprompt um weiterführende Themen von ChatGPT sortieren zu lassen. Die einzelnen Segmente werden automatisch ergänzt.

Systemprompt: *Schreibe 5 weiterführende Themenüberschriften, die zu diesen Textabschnitten passen. Sie sollen spezifisch auf eine Sache aus diesem Abschnitt abzielen, oder ein ähnliches Thema beschreiben. Jede Themenüberschrift soll aus 1-3 Wörtern Stichwörtern bestehen. Antworte in JSON Format als ein Array 'Themen' in welchem nur die Themen stehen*

Userprompt:

1: Keine falsche Assoziation. Peter der Große hat in seiner Jugend die Niederlande besucht. Amsterdam gefiel ihm. Petersburg zu bauen, war aber keine nostalgische Entscheidung, sondern hatte machtstrategische Gründe. Dem Zaren lag Moskau als Hauptstadt zu weit im Osten. Der Nordriese, wie Peter manchmal genannt wird, wollte sein Reich europäischer machen, auch durch den Bau einer neuen Hauptstadt.

2: Und das waren ja hochgebildete und bedeutsame Personen unter ihnen, die da für das holländische Leben eine große Rolle spielten. Zentrum der neu gegründeten Republik war Amsterdam. Das mächtige, prächtige Amsterdam. Die reichste Stadt der Welt. Metropole eines weltumspannenden Handelsimperiums und so multikulturell wie vor ihm nur das alte Rom gewesen war. Alle Pracht und alle Luxusgüter aus aller Herren Länder kamen nach Amsterdam.

3: Bis zu meinem vierten Lebensjahr wohnte ich in Frankfurt. Da wir Juden sind, ging dann mein Vater 1933 in die Niederlande. Amsterdam, das bedeutet unbeschwerte Kinderjahre. Bis die Nazis im Mai 1940 auch die Niederlande besetzen. Die Schlinge zieht sich zu für die Juden dort. Eine antisemitische Verordnung jagt die andere.

Beispielantwort:

```
{
    "Themen": [
        "Machtstrategie Peters",
        "Amsterdams Reichtum",
        "Holländisches Handelsimperium",
        "Jüdisches Exil",
        "Nazibesatzung Niederlande"
    ]
}
```

ChatGPT Segment-Grenzen finden

Ein Beispielprompt um einen zum Thema passenden wichtigen Ausschnitt aus einem Podcast zu generieren.

Systemprompt:

Schau dir das folgende Dokument an und wähle aus, welcher Abschnitt zur Beantwortung der Frage geeignet ist.

document

Antworte in JSON Format mit einem Text 'Begründung', der beschreibt warum dieser Textausschnitt dazu passt. Außerdem einen 'Start' für die Nummer des Satzes, ab dem der Abschnitt beginnt und einem 'Ende' für die Nummer des Satzes wo der Abschnitt aufhört. Wenn es keinen Abschnitt gibt der wirklich passt, antworte mit 'Start': 0 und 'Ende':0

ChatGPT Themengenerierung

Ein Beispielprompt um Themen zur Evaluation der Embedding-Modelle zu erstellen.

Generiere 30 verschiedene Suchthemen, die aus den Themenbereichen Geschichte, Naturwissenschaft, Gesellschaft oder Philosophie kommen und auch über die deutsche Kultur handeln. Die Themen sollen 1-3 verschiedene Wörter umfassen.

ChatGPT Fragengenerierung

Ein Beispielprompt um Fragen zur Evaluation der Embedding-Modelle zu erstellen.

Kannst du 30 Fragen generieren, wie „Wann werden Menschen auf dem Mars sein?“ oder „Wo wird urbaner Gartenbau betrieben?“

ChatGPT Evaluation

Ein Beispielprompt um Retrieval Ergebnisse zu evaluieren. temperature=0,5 seed=111

System-prompt:

Du bewertest die Ausgabe eines Text Retrieval Systems. Bewerte, wie gut die gefundenen Textabschnitte zu dem gestellten Thema passen. Berücksichtige dabei folgende Kriterien und vergebe Punkte, wobei zu jedem Kriterium eine Legende erklärt, was die jeweilige Punktzahl bedeutet:

- Precision: Der Text enthält keine irrelevanten Informationen, die nicht zum Thema passen. 0 Punkte: Der Text ist völlig irrelevant. 1-2 Punkte: Der Text enthält größtenteils irrelevante Informationen. 3-4 Punkte: Der Text ist überwiegend relevant mit einigen irrelevanten Details. 5 Punkte: Der Text ist vollständig relevant ohne irrelevante Informationen.

- Cohesion: Die einzelnen Textausschnitte passen inhaltlich zusammen. 0 Punkte: Kein Zusammenhang zwischen den Textausschnitten. 1 Punkt: Geringer Zusammenhang zwischen den Textausschnitten. 2 Punkte: Hoher Zusammenhang und gute inhaltliche Verknüpfung zwischen den Textausschnitten.

- Uniqueness: Die Textausschnitte wiederholen keine Informationen. 0 Punkte: Starke Wiederholungen von Informationen. 1-2 Punkte: Einige Wiederholungen, aber auch neue Informationen. 3 Punkte: Jeder Textausschnitt liefert einzigartige und neue Informationen.

Antworte in JSON Format, indem du zunächst eine Begründung "Reason"gibst, die aus einem Satz besteht, und bewerte dann jeweils mit einem Score für "Precision", "Cohesionünd Üniqueness".

User-Prompt:

Textabschnitte: documents

———

Frage: query

Literatur

[1] *1968 - Das Ausnahmejahr*. https://www.ardaudiothek.de/episode/radiowissen/1968-das-ausnahmejahr/bayern-2/78757544/. (Besucht am 14. 03. 2024).

[2] *Appropriate Uses For SQLite*. https://www.sqlite.org/whentouse.html. (Besucht am 14. 03. 2024).

[3] *ARD Mediathek erreicht täglich rund 2,3 Millionen Menschen*. https://www.swr.de/unternehmen/kommunikation/pressemeldungen/jahrespressegespraech-2024-ard-mediathek-ard-audiothek-100.html. März 2024. (Besucht am 14. 03. 2024).

[4] jeanine Banks und tris warkentin. *Gemma: Introducing New State-of-the-Art Open Models*. https://blog.google/technology/developers/gemma-open-models/. Feb. 2024. (Besucht am 14. 03. 2024).

[5] Y. Bengio, P. Simard und P. Frasconi. „Learning Long-Term Dependencies with Gradient Descent Is Difficult“. In: *IEEE Transactions on Neural Networks* 5.2 (März 1994), S. 157–166. ISSN: 1941-0093. DOI: 10.1109/72.279181. (Besucht am 15. 03. 2024).

[6] Tirza Biron u. a. „Automatic Detection of Prosodic Boundaries in Spontaneous Speech“. In: *PLOS ONE* 16.5 (Mai 2021), e0250969. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0250969. (Besucht am 11. 03. 2024).

[7] Tom Brown u. a. „Language Models Are Few-Shot Learners“. In: *Advances in Neural Information Processing Systems*. Bd. 33. Curran Associates, Inc., 2020, S. 1877–1901. (Besucht am 15. 03. 2024).

[8] chatgpt. *ChatGPT*. https://chat.openai.com. (Besucht am 14. 03. 2024).

[9] Juntian Chen, Yubo Tao und Hai Lin. „Visual Exploration and Comparison of Word Embeddings“. In: *Journal of Visual Languages & Computing* 48 (2018), S. 178–186. (Besucht am 14. 03. 2024).

[10] Kyunghyun Cho u. a. „On the Properties of Neural Machine Translation: Encoder–Decoder Approaches“. In: *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*. Doha, Qatar: Association for Computational Linguistics, 2014, S. 103–111. DOI: 10.3115/v1/W14-4012. (Besucht am 14. 03. 2024).

[11] Gobinda G. Chowdhury. *Introduction to Modern Information Retrieval*. Facet Publishing, 2010. ISBN: 978-1-85604-694-7.

[12] chroma. *Embeddings | Chroma*. https://www.trychroma.com/embeddings. (Besucht am 11. 03. 2024).

[13] claude. *Claude*. https://www.anthropic.com/claude. (Besucht am 14. 03. 2024).

[14] Luciano da Fontoura Costa und Gonzalo Travieso. *Fundamentals of Neural Networks: By Laurene Fausett. Prentice-Hall, 1994, Pp. 461, ISBN 0-13-334186-0*. 1996.

[15] Haixing Dai u. a. *SAMAug: Point Prompt Augmentation for Segment Anything Model*. Okt. 2023. arXiv: 2307.01187 [cs]. (Besucht am 14. 03. 2024).

[16] *Datatypes In SQLite*. https://www.sqlite.org/datatype3.html. (Besucht am 14. 03. 2024).

[17] Jacob Devlin u. a. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. Mai 2019. arXiv: 1810.04805 [cs]. (Besucht am 21. 11. 2023).

[18] Atanu Dey, Mamata Jenamani und Jitesh J. Thakkar. „Lexical TF-IDF: An n-Gram Feature Space for Cross-Domain Classification of Sentiment Reviews“. In: *Pattern Recognition and Machine Intelligence*. Hrsg. von B. Uma Shankar u. a. Cham: Springer International Publishing, 2017, S. 380–386. ISBN: 978-3-319-69900-4. DOI: 10.1007/978-3-319-69900-4_48.

[19] Andrew Duncan u. a. „Machine Learning-Based Early Warning System for Urban Flood Management“. In: University of Exeter, 2013. ISBN: 978-0-9926529-0-6. (Besucht am 15. 03. 2024).

[20] G. Dupret. „Discounted Cumulative Gain and User Decision Models“. In: (2011), S. 2–13. DOI: 10.1007/978-3-642-24583-1_2. (Besucht am 15. 03. 2024).

[21] *Embeddings*. https://docs.voyageai.com/docs/embeddings. (Besucht am 14. 03. 2024).

[22] Muhammed Erdem. *Muhammed/Mini-Player*. März 2024. (Besucht am 14. 03. 2024).

[23] eurovox. *EuroVOX*. https://tech.ebu.ch/eurovox. Jan. 2024. (Besucht am 12. 03. 2024).

[24] FFmpeg. *FFmpeg*. https://ffmpeg.org/. (Besucht am 08. 03. 2024).

[25] Firevince. *Firevince/Batchelorarbeit*. Feb. 2024. (Besucht am 15. 03. 2024).

[26] martin fowler. *Microservices*. https://martinfowler.com/articles/microservices.html. (Besucht am 14. 03. 2024).

[27] *Free German Dictionary*. https://sourceforge.net/projects/germandict/. Okt. 2021. (Besucht am 14. 03. 2024).

[28] gemini. *Gemini – chatten und inspirieren lassen*. https://gemini.google.com. (Besucht am 14. 03. 2024).

[29] *Giga Grünheide - Tesla in Brandenburg*. https://www.ardaudiothek.de/sendung/giga-gruenheide-tesla-in-brandenburg/80566220/. (Besucht am 14. 03. 2024).

[30] *Gunicorn - Python WSGI HTTP Server for UNIX*. https://gunicorn.org/. (Besucht am 14. 03. 2024).

[31] Donna K. Harman. *Overview of the Third Text REtrieval Conference (TREC-3)*. DIANE Publishing, 1995. ISBN: 978-0-7881-2945-2.

[32] Ying D He und Mehmet Kayaalp. „A Comparison of 13 Tokenizers on MEDLINE“. In: (2006). DOI: 10.13140/2.1.1133.1206. (Besucht am 13. 03. 2024).

[33] Sepp Hochreiter und Jürgen Schmidhuber. „Long Short-Term Memory“. In: *Neural computation* 9.8 (1997), S. 1735–1780. (Besucht am 13. 03. 2024).

[34] *Home | Chroma*. https://www.trychroma.com/. (Besucht am 14. 03. 2024).

[35] Matthew Honnibal und Ines Montani. „spaCy 2: Natural Language Understanding with Bloom Embeddings, Convolutional Neural Networks and Incremental Parsing“. In: *To appear* 7.1 (2017), S. 411–420.

[36] *Implementation Limits For SQLite*. https://www.sqlite.org/limits.html. (Besucht am 14. 03. 2024).

[37] TIOBE Index. „TIOBE Index—May 2023“. In: *Accessed: Jun* 2 (2023).

[38] *Jiaaro/Pydub @ GitHub*. https://pydub.com/. (Besucht am 14. 03. 2024).

[39] Albert Q. Jiang u. a. *Mixtral of Experts*. Jan. 2024. arXiv: 2401.04088 [cs]. (Besucht am 16. 02. 2024).

[40] Rosie Jones u. a. *TREC 2020 Podcasts Track Overview*. März 2021. DOI: 10.48550/arXiv.2103.15953. arXiv: 2103.15953 [cs]. (Besucht am 06. 11. 2023).

[41] *Journalistische Grundsätze*. https://www.ard.de/die-ard/Gemeinsame-journalistische-Grundsaetze-der-ARD-100. Feb. 2024. (Besucht am 14. 03. 2024).

[42] Myunghwa Kang und Ulrike Gretzel. „Effects of Podcast Tours on Tourist Experiences in a National Park“. In: *Tourism Management* 33.2 (Apr. 2012), S. 440–455. ISSN: 02615177. DOI: 10.1016/j.tourman.2011.05.005. (Besucht am 08. 11. 2023).

[43] Marzena Kryszkiewicz. „Determining Cosine Similarity Neighborhoods by Means of the Euclidean Distance“. In: *Rough Sets and Intelligent Systems - Professor Zdzisław Pawlak in Memoriam: Volume 2*. Hrsg. von Andrzej Skowron und Zbigniew Suraj. Berlin, Heidelberg: Springer, 2013, S. 323–345. ISBN: 978-3-642-30341-8. DOI: 10.1007/978-3-642-30341-8_17. (Besucht am 14. 03. 2024).

[44] Selva Kumar. *Semantic Search with ElasticSearch*. https://blog.gopenai.com/semantic-search-with-elasticsearch-1dab248d116f. Sep. 2023. (Besucht am 14. 03. 2024).

[45] Philippe Laban u. a. „NewsPod: Automatic and Interactive News Podcasts“. In: *27th International Conference on Intelligent User Interfaces*. Helsinki Finland: ACM, März 2022, S. 691–706. ISBN: 978-1-4503-9144-3. DOI: 10.1145/3490099.3511147. (Besucht am 06. 11. 2023).

[46] Thomas K. Landauer und Susan T. Dumais. „A Solution to Plato's Problem: The Latent Semantic Analysis Theory of Acquisition, Induction, and Representation of Knowledge.“ In: *Psychological review* 104.2 (1997), S. 211. (Besucht am 13. 03. 2024).

[47] Kenton Lee u. a. *End-to-End Neural Coreference Resolution*. Dez. 2017. arXiv: 1707.07045 [cs]. (Besucht am 14. 03. 2024).

[48] Yuchun Lee. „Handwritten Digit Recognition Using K Nearest-Neighbor, Radial-Basis Function, and Backpropagation Neural Networks“. In: *Neural Computation* 3.3 (Sep. 1991), S. 440–449. ISSN: 0899-7667. DOI: 10.1162/neco.1991.3.3.440. (Besucht am 15. 03. 2024).

[49] lemur. *Lemur Project Components: Indri*. https://www.lemurproject.org/indri.php. (Besucht am 09. 03. 2024).

[50] listennotes. *Podcast Stats: How Many Podcasts Are There?* https://www.listennotes.com/podcast-stats/. (Besucht am 02. 03. 2024).

[51] Yu A. Malkov und D.A. Yashunin. *Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs*. Apr. 2020. (Besucht am 13. 03. 2024).

[52] Dirk Maroni u. a. „KÜNSTLICHE INTELLIGENZ IM PRODUKTIVEN EINSATZ FÜR DIE AUTOMATISIERTE ERSCHLIESSUNG IM MULTIMEDIALEN PRODUKTIONSPROZESS“. In: (2020).

[53] Tomas Mikolov u. a. *Efficient Estimation of Word Representations in Vector Space*. Sep. 2013. arXiv: 1301.3781 [cs]. (Besucht am 13. 03. 2024).

[54] miniLM. *Sentence-Transformers/All-MiniLM-L6-v2 · Hugging Face*. https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2. März 2024. (Besucht am 13. 03. 2024).

[55] *Models - Hugging Face*. https://huggingface.co/models. Feb. 2024. (Besucht am 14. 03. 2024).

[56] mteb. *MTEB Leaderboard - a Hugging Face Space by Mteb*. https://huggingface.co/spaces/mteb/leaderboard. (Besucht am 11. 03. 2024).

[57] Muennighoff. *Mteb/Leaderboard · Restart the Space for New Models*. https://huggingface.co/spaces/mteb/leaderboard/discussions/28. Sep. 2023. (Besucht am 05. 03. 2024).

[58] Niklas Muennighoff u. a. *MTEB: Massive Text Embedding Benchmark*. März 2023. arXiv: 2210.07316 [cs]. (Besucht am 12. 01. 2024).

[59] Ben Naismith, Phoebe Mulcaire und Jill Burstein. „Automated Evaluation of Written Discourse Coherence Using GPT-4“. In: *Proceedings of the 18th Workshop on Innovative Use of NLP for Building Educational Applications (BEA 2023)*. Hrsg. von Ekaterina Kochmar u. a. Toronto, Canada: Association for Computational Linguistics, Juli 2023, S. 394–403. DOI: 10.18653/v1/2023.bea-1.32. (Besucht am 11. 03. 2024).

[60] Nic Newman u. a. „Reuters Institute Digital News Report 2022“. In: (2022).

[61] Andrew Ng. *Sequenz Models*. Coursera, 2023. (Besucht am 20. 10. 2023).

[62] Filippa Nilsson und Jonatan Tuvstedt. *GPT-4 as an Automatic Grader : The Accuracy of Grades Set by GPT-4 on Introductory Programming Assignments*. 2023. (Besucht am 11. 03. 2024).

[63] *Nutzungsbedingungen*. https://www.ardaudiothek.de/nutzungsbedingungen/. (Besucht am 14. 03. 2024).

[64] *OpenAI Platform*. https://platform.openai.com. (Besucht am 14. 03. 2024).

[65] *Our Next-Generation Model: Gemini 1.5*. https://blog.google/technology/ai/google-gemini-next-generation-model-february-2024/. Feb. 2024. (Besucht am 15. 03. 2024).

[66] Long Ouyang u. a. „Training Language Models to Follow Instructions with Human Feedback“. In: ().

[67] Lawrence Page u. a. „The PageRank Citation Ranking: Bringing Order to the Web“. In: ().

[68] Narendra Patwardhan, Stefano Marrone und Carlo Sansone. „Transformers in the Real World: A Survey on Nlp Applications“. In: *Information* 14.4 (2023), S. 242. (Besucht am 14. 03. 2024).

[69] Jeffrey Pennington, Richard Socher und Christopher Manning. „Glove: Global Vectors for Word Representation“. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, 2014, S. 1532–1543. DOI: 10.3115/v1/D14-1162. (Besucht am 21. 12. 2023).

[70] Matthew E. Peters u. a. *Deep Contextualized Word Representations*. März 2018. arXiv: 1802.05365 [cs]. (Besucht am 13. 03. 2024).

[71] pi. *Pi, Your Personal AI*. https://pi.ai/discover. (Besucht am 14. 03. 2024).

[72] pickle. *Pickle — Python Object Serialization*. https://docs.python.org/3/library/pickle.html. (Besucht am 11. 03. 2024).

[73] Muskaan Pirani u. a. „A Comparative Analysis of ARIMA, GRU, LSTM and BiLSTM on Financial Time Series Forecasting“. In: *2022 IEEE International Conference on Distributed Computing and Electrical Circuits and Electronics (ICDCECE)*. Ballari, India: IEEE, Apr. 2022, S. 1–6. ISBN: 978-1-66548-316-2. DOI: 10.1109/ICDCECE53908.2022.9793213. (Besucht am 06. 03. 2024).

[74] *Playground - https://api.ardaudiothek.de/graphql*. https://api.ardaudiothek.de/graphql. (Besucht am 14. 03. 2024).

[75] Alec Radford u. a. „Robust Speech Recognition via Large-Scale Weak Supervision“. In: ().

[76] *radioWissen*. https://www.ardaudiothek.de/sendung/radiowissen/5945518/. (Besucht am 14. 03. 2024).

[77] repodiac. *Repodiac/German_compound_splitter*. Juli 2023. (Besucht am 14. 03. 2024).

[78] Bayerischer Rundfunk. *Manuskripte : radioWissen*. https://www.br.de/radio/bayern2/service/manuskripte/radiowissen/radiowissen-manuskripte108.html. indexPage. Feb. 2024. (Besucht am 14. 03. 2024).

[79] *Salesforce/SFR-Embedding-Mistral · Hugging Face*. https://huggingface.co/Salesforce/SFR-Embedding-Mistral. (Besucht am 14. 03. 2024).

[80] KaShun Shum, Shizhe Diao und Tong Zhang. *Automatic Prompt Augmentation and Selection with Chain-of-Thought from Labeled Data*. Feb. 2024. arXiv: 2302.12822 [cs]. (Besucht am 14. 03. 2024).

[81] spacy. *German · spaCy Models Documentation*. https://spacy.io/models/de. 2024. (Besucht am 17. 02. 2024).

[82] Karen Sparck Jones. „A Statistical Interpretation of Term Specificity and Its Application in Retrieval“. In: *Journal of documentation* 28.1 (1972), S. 11–21. (Besucht am 13. 03. 2024).

[83] *SYSTRAN/Faster-Whisper*. SYSTRAN. März 2024. (Besucht am 14. 03. 2024).

[84] tiktoken. *Openai/Tiktoken*. OpenAI. März 2024. (Besucht am 11. 03. 2024).

[85] Anna Tong und Anna Tong. „Exclusive: ChatGPT Traffic Slips Again for Third Month in a Row“. In: *Reuters* (Sep. 2023). (Besucht am 12. 03. 2024).

[86] Hugo Touvron u. a. *Llama 2: Open Foundation and Fine-Tuned Chat Models*. Juli 2023. arXiv: 2307.09288 [cs]. (Besucht am 14. 03. 2024).

[87] Ashish Vaswani u. a. *Attention Is All You Need*. Aug. 2023. arXiv: 1706.03762 [cs]. (Besucht am 18. 11. 2023).

[88] Thomas Wolf u. a. *HuggingFace's Transformers: State-of-the-art Natural Language Processing*. Juli 2020. arXiv: 1910.03771 [cs]. (Besucht am 14. 03. 2024).

[89] Vilém Zouhar u. a. *A Formal Perspective on Byte-Pair Encoding*. Juni 2023. arXiv: 2306.16837 [cs, math]. (Besucht am 11. 03. 2024).

Glossar

API Application Programming Interface. i

GRU Gated Recurrent Unit. i

LLM Large language model. i

LSTM Long Short-Term Memory. i

MTEB Massive Text Embedding Benchmark. i

RNN Recurrent neural network. i