

Kategorisierung und Tagging von Ressourcen mithilfe von LLMs

Nihad Badalov

Betreuung: Michael Mayer · Abgabetermin: 30.04.2026

Abstract

Im Zeitalter der Informationsüberflutung stellt die langfristige Organisation großer Mengen an Informationen wie Dokumenten, Lernmaterialien und Notizen eine zunehmende Herausforderung dar. Mit wachsendem Umfang werden solche Informationssammlungen häufig unübersichtlich, was den gezielten Zugriff erschwert. Eine verbreitete Methode zur Strukturierung ist das Tagging, bei dem Informationen mit Schlagwörtern zur verbesserten Kategorisierung und Auffindbarkeit versehen werden. Klassische, auf Machine Learning basierende Tagging-Ansätze erweisen sich dabei oft als komplex und sind auf umfangreiche annotierte Trainingsdaten angewiesen, um qualitativ hochwertige Ergebnisse zu erzielen. Vor diesem Hintergrund untersucht diese Arbeit den Einsatz großer Sprachmodelle (Large Language Models, LLMs) und strukturierter Modellausgaben als alternativen Ansatz zur automatisierten Informationsorganisation.

Inhaltsverzeichnis

Inhaltsverzeichnis

1. Einleitung	3
1.1. Motivation	3
1.2. Problemstellung	3
1.3. Zielsetzung und Forschungsfrage	3
2. Grundlagen	4
2.1. Informationsretrieval und Metadaten	4
2.2. Tagging-Systeme und ihre Herausforderungen	4
2.3. Klassische Verfahren des Machine Learning	4
2.3.1. Bag-of-Words und TF-IDF	5
2.3.2. Naive Bayes	5
2.3.3. Support Vector Machines	5
2.4. Grenzen klassischer Ansätze	6
2.5. Large Language Models	6
2.5.1. Grundprinzipien und Training	6
2.5.2. Transformer-Architektur (konzeptionell)	6
2.5.3. Semantisches Verständnis und Kontext	7
2.5.4. Zero-Shot- und Few-Shot-Lernen	7
3. Methodik	7
3.1. Problemformulierung der automatisierten Verschlagwortung	7
3.2. Ansatz mit Large Language Models	7
3.3. Prompt-basierte Klassifikation	8
3.4. Mehrfachklassifikation (Multi-Label-Tagging)	8
3.5. Evaluationskriterien	8
4. Umsetzung	8
4.1. Datenerhebung	8
4.1.1. Datenquelle (Twitter)	8
4.1.2. Eigenschaften und Herausforderungen der Daten	9
4.2. Datenverarbeitungspipeline	9
4.2.1. Extraktion	10
4.2.2. Vorverarbeitung	10
4.2.3. Klassifikation	11
4.2.4. Speicherung der Tags	11
4.3. Implementierungsdetails	11
4.3.1. Projektstruktur	11
4.3.2. Klassifikationslogik	12
4.3.3. Scraper-Implementierung	12
Literaturverzeichnis	12

1. Einleitung

1.1. Motivation

Digitale Wissensarbeit ist heute durch eine hohe Verfügbarkeit und gleichzeitige Flüchtigkeit von Informationen geprägt. Fachartikel, Blogbeiträge, Dokumentationen, Social-Media-Beiträge und persönliche Notizen werden fortlaufend gesammelt, weiterverarbeitet und in unterschiedliche Werkzeuge übernommen. Mit wachsender Menge steigt jedoch nicht automatisch die Nutzbarkeit dieser Informationen. Ohne eine konsistente Struktur entstehen Sammlungen, in denen relevante Inhalte zwar vorhanden sind, später aber nur schwer wiedergefunden werden. Genau an dieser Stelle gewinnt die Verschlagwortung von Ressourcen an Bedeutung, weil sie Informationen nicht nur speichert, sondern auch inhaltlich erschließbar macht.

Im persönlichen wie im akademischen Kontext wird Tagging häufig dennoch manuell durchgeführt. Diese Vorgehensweise ist aufwendig, fehleranfällig und stark von der jeweiligen Person abhängig. Unterschiedliche Begriffe, wechselnde Granularität und uneinheitliche Benennungen führen dazu, dass ähnliche Ressourcen nicht unter denselben Schlagwörtern abgelegt werden. Besonders problematisch wird dies bei kurzen, kontextreichen Quellen wie Beiträgen auf X beziehungsweise Twitter, deren Aussage oft implizit bleibt und ohne inhaltliche Einordnung schwer interpretierbar ist. Die Motivation dieser Arbeit besteht deshalb darin, zu untersuchen, ob sich große Sprachmodelle für eine automatisierte, zugleich aber semantisch brauchbare Verschlagwortung solcher Ressourcen eignen.

1.2. Problemstellung

Aus informationswissenschaftlicher Sicht handelt es sich bei der automatisierten Verschlagwortung um ein Klassifikationsproblem, das mehrere Schwierigkeiten gleichzeitig vereint. Erstens liegt häufig keine große, sauber annotierte Trainingsmenge vor. Zweitens sind die zu klassifizierenden Texte oft kurz, unvollständig oder sprachlich informell. Drittens sollen die vergebenen Tags nicht flach, sondern in einer hierarchischen Struktur organisiert werden, damit sowohl grobe als auch spezifische Themenbereiche abbildbar sind. Eine Ressource kann zudem mehreren Kategorien gleichzeitig zugeordnet werden, wodurch ein Multi-Label-Szenario entsteht [1].

Klassische Verfahren des Machine Learning erzielen in klar umrissenen Anwendungsfällen zwar gute Ergebnisse, erfordern aber typischerweise eine vorgängige Merkmalsrepräsentation, Trainingsdaten und eine stabile Problemdefinition [2]. Im vorliegenden Projekt liegt die Herausforderung anders: Es existiert bereits ein hierarchischer Tag-Baum, doch es fehlt an umfangreich annotierten Beispielen für das Training eines speziellen Modells. Gleichzeitig soll das System in der Lage sein, auch für neue oder nur teilweise passende Inhalte sinnvolle Vorschläge zu machen. Daraus ergibt sich die zentrale Problemstellung, wie eine praktische, robuste und nachvollziehbare automatische Verschlagwortung unter realen Projektbedingungen umgesetzt werden kann.

1.3. Zielsetzung und Forschungsfrage

Ziel dieser Facharbeit ist die theoretische Einordnung und praktische Umsetzung eines Prototyps zur automatisierten Kategorisierung digitaler Ressourcen mithilfe großer Sprachmodelle. Im Mittelpunkt steht nicht der Vergleich zahlreicher Modellvarianten, sondern die Entwicklung einer funktionsfähigen Verarbeitungspipeline, die Ressourcen aus einer Eingabeliste entgegennimmt, deren Inhalte extrahiert, einen strukturierten Klassifikationsaufruf ausführt und die Ergebnisse in einer Datenbank speichert. Der konkrete Prototyp dieses Projekts verarbeitet derzeit Beiträge von X beziehungsweise Twitter, ist in Rust implementiert und nutzt ein extern aufgerufenes LLM zur Zuordnung in eine vorgegebene Tag-Hierarchie.

Die leitende Forschungsfrage lautet daher: Inwieweit eignen sich Large Language Models dazu, kurze digitale Ressourcen ohne spezielles domänenspezifisches Training automatisch in ein hierarchisches Multi-Label-Tagging-System einzuordnen? Daraus ergeben sich zwei untergeordnete Fragen. Erstens ist zu klären, welche fachlichen Grenzen klassische Verfahren im gegebenen Szenario aufweisen. Zweitens soll untersucht werden, wie eine konkrete Implementierung beschaffen sein muss, damit die LLM-basierte Klassifikation technisch zuverlässig und praktisch nutzbar wird.

2. Grundlagen

2.1. Informationsretrieval und Metadaten

Informationsretrieval bezeichnet den Bereich der Informatik und Informationswissenschaft, der sich mit der Speicherung, Strukturierung, Suche und Wiederauffindung von Informationen beschäftigt. Anders als bei klassischen Datenbanksystemen stehen dabei häufig un- oder halbstrukturierte Inhalte im Mittelpunkt, etwa Texte, Webseiten oder Dokumente. Ziel ist es, aus einer größeren Informationsmenge diejenigen Elemente zu identifizieren, die für eine Anfrage oder einen Nutzungskontext relevant sind [2]. Für die Qualität eines Retrieval-Systems ist daher nicht allein entscheidend, ob Inhalte vorhanden sind, sondern ob sie in einer Form beschrieben werden, die ihre spätere Auffindbarkeit unterstützt.

Metadaten übernehmen in diesem Zusammenhang eine zentrale Rolle. Sie beschreiben Ressourcen durch zusätzliche Merkmale wie Titel, Autor, Thema, Entstehungszeitpunkt oder Schlagwörter. Solche beschreibenden Informationen schaffen eine zweite Ebene über dem eigentlichen Inhalt und ermöglichen es, Dokumente nicht nur wortwörtlich, sondern auch semantisch oder organisatorisch zu erschließen. In digitalen Wissenssammlungen sind Tags eine besonders flexible Form solcher Metadaten: Sie können thematische Zuordnungen herstellen, Querverbindungen sichtbar machen und den Übergang zwischen freier Sammlung und systematischer Ordnung erleichtern.

Gerade für heterogene Ressourcensammlungen sind Metadaten wichtig, weil Inhalte oft aus sehr unterschiedlichen Quellen stammen. Während ein wissenschaftlicher Artikel über umfangreiche bibliographische Angaben verfügt, enthält ein kurzer Social-Media-Beitrag zunächst nur wenige strukturierte Felder. Umso wichtiger ist es, aus den vorhandenen Informationen eine abstrahierte thematische Beschreibung abzuleiten. Die automatische Vergabe solcher Beschreibungen ist damit ein naheliegender Anwendungsfall des Informationsretrievals und zugleich eine Voraussetzung für spätere Such-, Filter- und Exportfunktionen.

2.2. Tagging-Systeme und ihre Herausforderungen

Tagging-Systeme ordnen Ressourcen Schlagwörter zu, um sie thematisch zu gruppieren und leichter auffindbar zu machen. Im Unterschied zu streng kontrollierten Vokabularen oder Taxonomien sind Tags oft flexibel, offen und nutzergetrieben. Diese Offenheit ist praktisch, weil neue Begriffe schnell eingeführt werden können und Nutzerinnen und Nutzer ihre eigene Sicht auf Inhalte abbilden können. Gleichzeitig entstehen dadurch typische Probleme, etwa uneinheitliche Schreibweisen, Synonyme, Mehrdeutigkeiten und stark schwankende Detailgrade [3].

Eine Ressource über Programmiersprachen könnte beispielsweise mit `rust`, `systems_programming`, `compiler`, `memory` oder `performance` markiert werden. Jede dieser Bezeichnungen ist potenziell sinnvoll, doch nicht jede repräsentiert dieselbe Ebene oder denselben Fokus. Ohne Strukturierung entstehen Sammlungen, in denen Tags nebeneinanderstehen, obwohl sie hierarchisch oder semantisch miteinander verbunden sind. Für Retrieval-Aufgaben ist das problematisch, weil die Suche entweder zu breit oder zu eng ausfallen kann.

Im vorliegenden Projekt wird deshalb kein flaches Tagging verwendet, sondern ein hierarchischer Tag-Baum. Dieser verbindet die Flexibilität einzelner Schlagwörter mit der Ordnungsfunktion einer Taxonomie. Ein Tag wie `cs/programming_languages/rust` ist aussagekräftiger als ein isoliertes `rust`, weil es den Begriff in einen größeren Kontext einordnet. Die Herausforderung verschiebt sich dadurch jedoch: Das System muss nicht nur ein passendes Thema erkennen, sondern auch die richtige Ebene innerhalb der Hierarchie wählen.

2.3. Klassische Verfahren des Machine Learning

Vor dem Aufkommen großer Sprachmodelle wurde die automatische Textklassifikation überwiegend mit klassischen Verfahren des Machine Learning umgesetzt. Diese Methoden basieren meist auf einer expliziten Merkmalsdarstellung von Texten, etwa durch Worthäufigkeiten, und einem darauf trainierten Klassifikator. Sie sind rechnerisch oft effizient und in gut definierten Szenarien leistungsfähig. Ihr Erfolg

hängt aber stark davon ab, wie gut Texte vorverarbeitet, Merkmale ausgewählt und Trainingsdaten gelabelt wurden [2].

Für das Verständnis der späteren LLM-basierten Lösung ist ein kurzer Blick auf die zentralen klassischen Ansätze sinnvoll. Besonders relevant sind Bag-of-Words- und TF-IDF-Repräsentationen als Grundlage, Naive Bayes als probabilistisches Verfahren und Support Vector Machines als starke lineare Klassifikatoren für hochdimensionale Textdaten.

2.3.1. Bag-of-Words und TF-IDF

Beim Bag-of-Words-Ansatz wird ein Text als ungeordnete Menge von Wörtern modelliert. Entscheidend ist dabei nicht die genaue Wortreihenfolge, sondern welche Terme vorkommen und mit welcher Häufigkeit. Auf diese Weise lässt sich jeder Text in einen Vektor überführen, dessen Dimensionen einzelnen Wörtern oder Token entsprechen. Das Verfahren ist konzeptionell einfach und für viele frühe Textklassifikationssysteme grundlegend gewesen [2].

Eine wichtige Verfeinerung stellt TF-IDF dar, also die Gewichtung durch Term Frequency und Inverse Document Frequency. Häufige Begriffe innerhalb eines Dokuments werden stärker gewichtet, während Wörter, die in nahezu allen Dokumenten vorkommen, an Bedeutung verlieren. Dadurch wird die Repräsentation informativer, weil charakteristische Begriffe stärker hervortreten. Salton und Buckley zeigen, dass unterschiedliche Gewichtungsschemata erheblichen Einfluss auf Retrieval- und Klassifikationsergebnisse haben können [4].

Trotz ihres Nutzens bleibt die Repräsentation jedoch überwiegend oberflächenorientiert. Zwei inhaltlich ähnliche Texte können sehr unterschiedlich erscheinen, wenn sie unterschiedliche Vokabeln verwenden. Umgekehrt können Texte mit ähnlichen Begriffen thematisch verschieden sein. Genau diese Begrenzung wird bei kurzen und kontextabhängigen Ressourcen besonders sichtbar.

2.3.2. Naive Bayes

Naive Bayes ist ein probabilistisches Klassifikationsverfahren, das auf dem Satz von Bayes beruht und die Merkmale eines Dokuments als bedingt unabhängig voneinander annimmt. Diese Annahme ist in realen Texten zwar stark vereinfacht, führt in der Praxis jedoch häufig zu brauchbaren und robusten Ergebnissen. Vor allem in frühen Anwendungen der Dokumentklassifikation war Naive Bayes wegen seiner Einfachheit und Effizienz weit verbreitet [2].

Für Textdaten existieren unterschiedliche Ereignismodelle, insbesondere multinomiale und Bernoulli-Varianten. McCallum und Nigam zeigen, dass die Wahl des Ereignismodells die Klassifikationsleistung deutlich beeinflussen kann und dokumentieren die Eignung von Naive Bayes für Textklassifikation trotz der starken Unabhängigkeitsannahme [5]. Die Stärke des Verfahrens liegt darin, mit vergleichsweise wenig Rechenaufwand Wahrscheinlichkeiten für Klassen zu schätzen.

Für komplexe Tagging-Systeme stößt Naive Bayes jedoch an Grenzen. Die Methode modelliert keine tieferen semantischen Beziehungen zwischen Begriffen und bildet Kontext nur indirekt über Worthäufigkeiten ab. Gerade bei mehrdeutigen oder sehr kurzen Texten kann das zu unscharfen Zuordnungen führen.

2.3.3. Support Vector Machines

Support Vector Machines, kurz SVM, gehören zu den klassischen Verfahren, die für Textklassifikation lange als besonders leistungsfähig galten. Ihr Grundprinzip besteht darin, im Merkmalsraum eine trennende Hyperebene zu finden, die Klassen möglichst gut voneinander separiert. Gerade in hochdimensionalen, spärlich besetzten Textvektoren funktioniert dieser Ansatz oft zuverlässig, weil viele Dokumente trotz großer Vokabularräume linear trennbar oder nahezu trennbar sind [6].

Für Textklassifikation ist die Methode deshalb attraktiv, weil sie mit TF-IDF- oder ähnlichen Repräsentationen gut kombinierbar ist. Joachims zeigt, dass SVMs in diesem Bereich hohe Genauigkeit erreichen und insbesondere bei großen Merkmalsräumen robust arbeiten [6]. Im Vergleich zu einfacheren Verfahren wie Naive Bayes ist das Modell jedoch schwerer interpretierbar und erfordert ebenfalls annotierte Trainingsdaten.

Auch SVMs lösen nicht das Grundproblem fehlender Semantik. Sie trennen Klassen auf Basis der vorgegebenen Merkmalsdarstellung, erzeugen diese Darstellung aber nicht selbst. Wenn relevante Inhalte in einem Text implizit bleiben oder nur durch Weltwissen erschließbar sind, kann auch ein starker Klassifikator nur begrenzt helfen.

2.4. Grenzen klassischer Ansätze

Die beschriebenen Verfahren haben wesentlich zur Entwicklung moderner Textklassifikation beigetragen, doch ihre Grenzen werden im Kontext dieser Arbeit deutlich. Erstens setzen sie in der Regel ein gelabeltes Korpus voraus. Für ein persönliches oder projektbezogenes Tagging-System liegt ein solches Korpus häufig nicht vor, und seine Erstellung wäre zeitaufwendig. Zweitens hängen klassische Verfahren stark von der Qualität der Merkmalsrepräsentation ab. Werden Synonyme, Mehrdeutigkeiten oder thematische Beziehungen nicht bereits in den Merkmalen erfasst, kann das Modell sie kaum nachträglich rekonstruieren [2].

Hinzu kommt, dass kurze Social-Media-Texte oft nur bruchstückhafte Informationen enthalten. Ein Beitrag kann auf eine Debatte verweisen, implizites Fachwissen voraussetzen oder wesentliche Teile seiner Bedeutung aus Links, Autorenschaft und Diskurskontext beziehen. Ein Bag-of-Words-Modell sieht in einem solchen Fall nur wenige Terme. Für die Auswahl eines spezifischen Tags innerhalb einer Hierarchie ist dies oft nicht ausreichend.

Schließlich ist das Ziel dieses Projekts nicht die Vorhersage genau einer Klasse, sondern die Zuweisung mehrerer möglichst spezifischer Tags in einer hierarchischen Struktur. Klassische Modelle können zwar für Multi-Label- oder Hierarchieaufgaben erweitert werden, doch dies erhöht Modellierungsaufwand und Komplexität. Für ein prototypisches System ohne große Trainingsdatenbasis liegt daher ein anderer Ansatz näher: die Nutzung eines bereits breit vortrainierten Sprachmodells.

2.5. Large Language Models

2.5.1. Grundprinzipien und Training

Large Language Models sind neuronale Sprachmodelle, die auf sehr großen Textmengen vortrainiert werden und auf dieser Grundlage statistische Regularitäten von Sprache, Wissen und typischen Textmustern erfassen. Im Kern lernen sie, welches Token mit welcher Wahrscheinlichkeit auf einen gegebenen Kontext folgt. Durch die enorme Menge an Trainingsdaten und Parametern entstehen Modelle, die nicht nur lokale Wortfolgen, sondern auch komplexere sprachliche und semantische Zusammenhänge verarbeiten können [7].

Der entscheidende Unterschied zu klassischen Textklassifikationsverfahren liegt darin, dass LLMs nicht speziell für eine einzelne Klassifikationsaufgabe trainiert werden müssen, um nützliche Ergebnisse zu liefern. Vielmehr können sie durch Anweisungen im Prompt auf neue Aufgaben ausgerichtet werden. Diese Fähigkeit macht sie insbesondere für Szenarien interessant, in denen keine große, sauber annotierte Trainingsmenge vorhanden ist, aber dennoch eine inhaltlich differenzierte Beurteilung nötig ist.

2.5.2. Transformer-Architektur (konzeptionell)

Die gegenwärtige Leistungsfähigkeit großer Sprachmodelle ist eng mit der Transformer-Architektur verbunden. Vaswani et al. führen mit dem Transformer ein Modell ein, das auf Self-Attention basiert und dadurch Abhängigkeiten zwischen Wörtern eines Textes parallel und kontextsensitiv verarbeiten kann [8]. Anstatt Wörter nur sequenziell zu betrachten, gewichtet das Modell, welche Teile des Eingabetextes für die Interpretation eines bestimmten Tokens besonders relevant sind.

Konzeptionell ist diese Architektur für Textklassifikation deshalb bedeutsam, weil sie Beziehungen über größere Distanzen hinweg erfassen kann. Wenn in einem Text erst spät klar wird, worauf sich ein Begriff bezieht, kann ein Transformer diese Information besser einbeziehen als klassische Modelle mit starren Vektorrepräsentationen. Für die hier betrachtete Aufgabe bedeutet das, dass nicht nur einzelne Schlüsselwörter, sondern auch ihre Einbettung in den Gesamtzusammenhang berücksichtigt werden können.

2.5.3. Semantisches Verständnis und Kontext

Im Zusammenhang mit LLMs wird häufig von semantischem Verständnis gesprochen. Fachlich präziser ist es, von kontextsensitiver Bedeutungsverarbeitung zu sprechen. Das Modell besitzt kein menschliches Verstehen, kann aber auf Basis seines Trainings sehr viele sprachliche Muster, thematische Bezüge und typische Diskurszusammenhänge in seine Vorhersagen einfließen lassen [7]. Für praktische Aufgaben wie Tagging ist genau diese Fähigkeit entscheidend, weil sie über rein oberflächenbasierte Schlüsselwörterkennung hinausgeht.

Ein kurzer Text über Compilerbau kann beispielsweise Begriffe wie `intermediate representation`, `optimization passes` oder `toolchain` enthalten, ohne das Oberthema explizit zu benennen. Ein LLM kann solche Signale in Beziehung setzen und daraus eine spezifische thematische Einordnung ableiten. Gerade in technischen Wissenssammlungen ist diese Eigenschaft wertvoll, weil viele Ressourcen nur für Personen mit Vorwissen unmittelbar eindeutig sind.

2.5.4. Zero-Shot- und Few-Shot-Lernen

Ein weiterer wichtiger Aspekt ist die Fähigkeit großer Sprachmodelle zum Zero-Shot- und Few-Shot-Lernen. Brown et al. zeigen am Beispiel von GPT-3, dass große Sprachmodelle neue Aufgaben allein auf Basis sprachlicher Instruktionen und weniger oder sogar ganz ohne Beispiele bearbeiten können [9]. Das Modell muss dabei nicht durch Gradientenupdates auf die konkrete Aufgabe nachtrainiert werden, sondern reagiert direkt auf die Formulierung des Prompts.

Für das vorliegende Projekt ist insbesondere Zero-Shot-Lernen relevant. Die Klassifikation erfolgt nicht mithilfe eines separat trainierten Tagging-Modells, sondern durch einen Prompt, der die Tag-Hierarchie, die Ressource und das gewünschte Ausgabeformat enthält. Wenige oder gar keine Beispiele im Prompt senken den Vorbereitungsaufwand und machen das System flexibel. Gleichzeitig steigt damit die Bedeutung einer präzisen Aufgabenbeschreibung, weil die Qualität der Ausgabe stark von der Formulierung des Prompts abhängt.

3. Methodik

3.1. Problemformulierung der automatisierten Verschlagwortung

Methodisch lässt sich das in dieser Arbeit behandelte Problem als hierarchische Multi-Label-Textklassifikation formulieren. Gegeben ist eine digitale Ressource, im aktuellen Prototyp repräsentiert durch eine URL zu einem X-Beitrag sowie den daraus extrahierten Textinhalt. Gesucht ist eine Menge von einem bis drei möglichst spezifischen Tags aus einem vorgegebenen Tag-Baum. Die Tags sollen thematisch passen, hierarchisch korrekt eingeordnet sein und die Ressource so beschreiben, dass sie später wiedergefunden und in Beziehung zu anderen Ressourcen gesetzt werden kann.

Im Unterschied zu einer einfachen Ein-Klassen-Klassifikation müssen mehrere Bedingungen gleichzeitig erfüllt werden. Eine Ressource kann mehreren Themenbereichen zugeordnet sein, etwa `distributed_systems` und `databases`. Gleichzeitig soll das System eher spezifische Blattknoten als sehr allgemeine Oberkategorien wählen. Wenn kein passender Tag existiert, soll das System außerdem einen begründeten Vorschlag für einen neuen Tag liefern. Die Methodik umfasst somit nicht nur Zuordnung, sondern auch kontrollierte Erweiterbarkeit der Tag-Struktur.

3.2. Ansatz mit Large Language Models

Der gewählte Ansatz nutzt ein Large Language Model als generischen Klassifikator, der ohne projektspezifisches Training auskommt. Statt ein Modell mit annotierten Beispielen auf den vorhandenen Tag-Baum zu trainieren, wird die Aufgabe zur Laufzeit in natürlicher Sprache formuliert. Das Modell erhält die aktuelle Tag-Hierarchie, die textuelle Beschreibung der Ressource und klare Regeln zur Ausgabe. Aus methodischer Sicht handelt es sich damit um eine instruktionale Zero-Shot-Klassifikation.

Dieser Ansatz ist für das Projekt aus drei Gründen sinnvoll. Erstens entfällt die aufwendige Erstellung eines Trainingsdatensatzes. Zweitens kann das Modell Weltwissen und semantische Relationen nutzen, um auch kurze oder indirekte Texte einzuordnen. Drittens lässt sich das System leicht anpassen, indem

der Tag-Baum oder die Prompt-Regeln verändert werden, ohne dass ein neues Modell trainiert werden muss. Die methodische Flexibilität ist damit höher als bei einem klassisch trainierten Spezialmodell.

3.3. Prompt-basierte Klassifikation

Die Klassifikation wird durch einen strukturierten Prompt gesteuert, der dem Sprachmodell die Aufgabe explizit vorgibt. Im Prompt enthält dieser Prompt erstens eine Rollenbeschreibung als Resource Classifier, zweitens Regeln zur Auswahl möglichst spezifischer Tags, drittens die aktuelle Tag-Hierarchie und viertens eine formatierte Ressourcendarstellung. Zusätzlich wird ein JSON-Ausgabeformat vorgegeben, das die Felder `tags`, `confidence`, `new_tags` und `reasoning` enthält. Der Prompt reduziert damit die Offenheit der Modellantwort und fördert eine maschinenlesbare Ausgabe.

Methodisch ist diese Form der Prompt-Gestaltung zentral, weil LLMs zwar flexibel, aber nicht deterministisch sind. Ein ungenauer Prompt könnte zu freien Textantworten, zu allgemeinen Kategorisierungen oder zu uneinheitlichen Formaten führen. Deshalb wird die Ausgabe auf ein klar beschriebenes JSON-Schema beschränkt. Die spätere Verarbeitung im Programm setzt nämlich voraus, dass die Modellantwort von `serde` geparkt werden kann. Die Prompt-basierte Klassifikation ist in diesem Projekt somit nicht nur eine sprachliche Instruktion, sondern Teil des technischen Schnittstellendesigns.

3.4. Mehrfachklassifikation (Multi-Label-Tagging)

Ein zentrales methodisches Merkmal des Systems ist die Mehrfachklassifikation. Viele technische Ressourcen behandeln mehrere Themen gleichzeitig. Ein Beitrag über eine Datenbank-Engine kann sich zugleich auf Architekturentscheidungen, Persistenz, Performance und Nebenläufigkeit beziehen. Eine Ein-Klassen-Entscheidung würde diese Mehrdimensionalität unzulässig reduzieren. Das Modell wird daher angewiesen, ein bis drei passende Tags zurückzugeben.

Damit die Mehrfachklassifikation nicht zu unspezifischen Sammel Listen führt, werden pro Tag zusätzlich Konfidenzwerte erwartet. Diese Werte sind im Prototyp keine mathematisch kalibrierten Wahrscheinlichkeiten, sondern modellgenerierte Einschätzungen, die als heuristische Stärke der Zuordnung interpretiert werden. In der Anwendung dienen sie dazu, besonders unsichere Fälle zu markieren. Zusätzlich sieht das Schema vor, neue Tags mit Elternkategorie und Begründung vorzuschlagen, wenn der bestehende Baum keine zufriedenstellende Zuordnung erlaubt.

3.5. Evaluationskriterien

Auch wenn in dieser Fassung noch keine systematische Ergebnisdiskussion vorgenommen wird, lassen sich klare Evaluationskriterien formulieren. Ein erstes Kriterium ist die inhaltliche Angemessenheit: Die vergebenen Tags sollen den Gegenstand einer Ressource korrekt und nicht nur oberflächlich beschreiben. Ein zweites Kriterium ist die Spezifität. Ein System, das systematisch nur allgemeine Oberkategorien wie `cs` oder `software_development` auswählt, wäre praktisch wenig nützlich, selbst wenn die Zuordnungen formal nicht falsch wären.

Ein drittes Kriterium betrifft die hierarchische Konsistenz. Die vorgeschlagenen Tags müssen in die bestehende Struktur passen und dürfen keine semantisch widersprüchlichen Kombinationen erzeugen. Viertens ist die Robustheit der Ausgabe wichtig: Für die technische Nutzbarkeit muss die Antwort zuverlässig im erwarteten JSON-Format vorliegen. Fünftens ist die praktische Anschlussfähigkeit zu nennen. Ein brauchbares System muss seine Ergebnisse speichern, exportieren und bei wiederholter Verarbeitung konsistent behandeln können. Diese Kriterien verbinden damit semantische, strukturelle und technische Qualitätsanforderungen.

4. Umsetzung

4.1. Datenerhebung

4.1.1. Datenquelle (Twitter)

Die Datenerhebung des vorliegenden Prototyps basiert auf einer einfachen, kontrollierbaren Eingabequelle: einer Datei mit URLs namens `test-classification-list`. Jede Zeile dieser Datei enthält eine Ressource, die klassifiziert werden soll. Im aktuellen Entwicklungsstand werden ausschließlich URLs von X

beziehungsweise Twitter verarbeitet. Die Entscheidung für diese Datenquelle ist pragmatisch begründet. Beiträge auf X sind leicht verlinkbar, technisch klar identifizierbar und zugleich inhaltlich anspruchsvoll, weil sie kurz, kontextreich und oft stark verdichtet formuliert sind.

Die Wahl dieser Quelle passt zum Ziel der Arbeit, ein praxisnahes System für persönliche Wissenssammlungen zu untersuchen. Viele technisch orientierte Beiträge, Diskussionen oder Hinweise auf Werkzeuge und Fachthemen werden heute in Social-Media-Form verbreitet. Solche Ressourcen gehen ohne systematische Organisation leicht verloren. Der Prototyp nutzt X-Beiträge daher als realistische Testumgebung für ein später erweiterbares Klassifikationssystem.

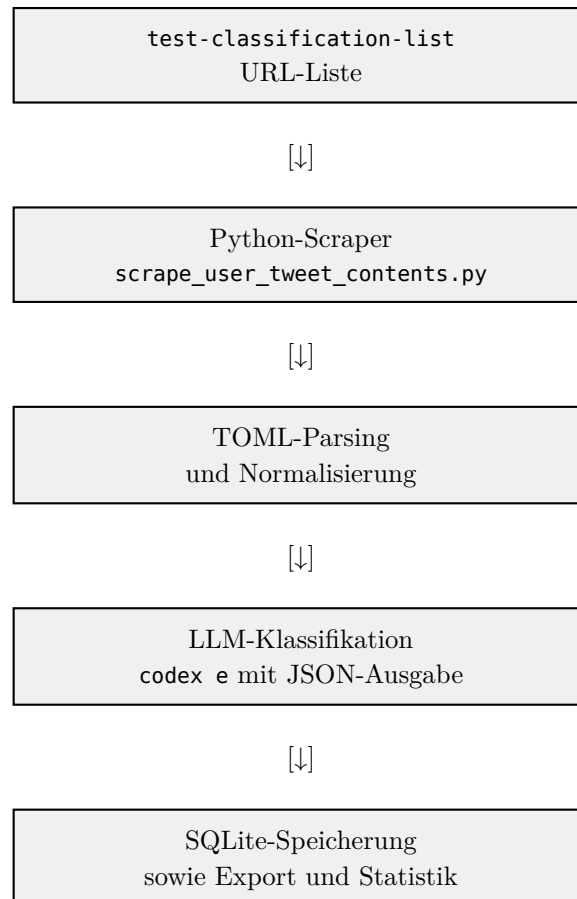
4.1.2. Eigenschaften und Herausforderungen der Daten

Die verwendeten Daten weisen mehrere Eigenschaften auf, die ihre automatische Verarbeitung erschweren. Zunächst sind die Texte kurz. Anders als wissenschaftliche Artikel oder längere Blogbeiträge enthalten Tweets meist nur wenige Sätze. Relevante Informationen bleiben häufig implizit oder sind auf externe Links ausgelagert. Zusätzlich wird in Social-Media-Beiträgen oft informelle Sprache verwendet, einschließlich Abkürzungen, Fachjargon, Ironie oder verkürzter Formulierungen.

Hinzu kommen strukturelle Herausforderungen. Die Eingabeliste kann Duplikate enthalten, was im Testdatensatz tatsächlich vorkommt. Außerdem ist ein Beitrag nicht nur durch seinen Text charakterisiert, sondern auch durch Metadaten wie Autor, Beitrag-ID und eingebettete URLs. Der Prototyp verarbeitet daher nicht allein den reinen Text, sondern verbindet ihn mit einer knappen kontextgebenden Titelform. Aus methodischer Sicht wird dadurch versucht, den Informationsverlust kurzer Texte teilweise auszugleichen, ohne den Aufwand eines vollständigen Diskurskontextes übernehmen zu müssen.

4.2. Datenverarbeitungspipeline

Die Verarbeitung der Ressourcen erfolgt in einer mehrstufigen Pipeline, die in [Listing 1] schematisch dargestellt ist. Ausgangspunkt ist eine Liste von URLs. Nach der Quelle-Erkennung wird der jeweilige Beitrag extrahiert, in eine einheitliche Textrepräsentation überführt, durch das Sprachmodell klassifiziert und schließlich zusammen mit seinen Tag-Zuordnungen in einer SQLite-Datenbank gespeichert. Zusätzlich unterstützt das System einen Export in JSON sowie eine einfache Statistikabfrage über den aktuellen Datenbestand.



Listing 1: Vereinfachte Datenverarbeitungspipeline des implementierten Prototyps.

4.2.1. Extraktion

Die eigentliche Verarbeitung beginnt in `src/main.rs`. Dort wird zunächst die Eingabedatei eingelesen und zeilenweise verarbeitet. Für jede URL bestimmt die Funktion `determine_resource_source`, ob es sich um eine unterstützte Quelle handelt. Gegenwärtig werden nur Adressen erkannt, die `twitter.com` oder `x.com` enthalten. Alle anderen Quellen werden mit einem Hinweis übersprungen. Diese Beschränkung macht deutlich, dass der Prototyp noch kein allgemeiner Web-Scraper ist, sondern zunächst einen klar abgegrenzten Ressourcentyp implementiert.

Für Twitter- beziehungsweise X-Links ruft das Programm die Funktion `scrapers::twitter::scrape(url)` auf. Diese extrahiert aus der URL die Tweet-ID und startet anschließend über `Command::new("python")` das externe Skript `scrape_user_tweet_contents.py`. Wenn die Extraktion erfolgreich war, liegt im Verzeichnis `scraped-tweets/` eine TOML-Datei mit dem Namen `tweet-<id>.toml` vor. Diese Datei dient als Übergabeformat zwischen Scraper und weiterer Rust-Verarbeitung. Die Extraktion ist damit technisch ausgelagert, aber sauber in die Hauptpipeline integriert.

4.2.2. Vorverarbeitung

Nach der Extraktion wird die erzeugte TOML-Datei mit `parse_scraped_tweet` eingelesen. Die Implementierung versucht zunächst, das erwartete Format mittels `serde` in eine interne Struktur zu deserialisieren. Dabei werden Felder wie `id`, `full_text` und der Autorenname unter `author.screen_name` berücksichtigt. Falls diese direkte Deserialisierung scheitert, greift eine Fallback-Routine, die relevante Zeilen manuell aus dem TOML-Text extrahiert. Diese zweistufige Verarbeitung erhöht die Robustheit gegenüber kleineren Formatabweichungen der Scraper-Ausgabe.

Das Ergebnis der Vorverarbeitung ist ein vereinheitlichtes Objekt vom Typ `ScrapedTweet`, das die drei wesentlichen Informationen `id`, `text` und `author` enthält. Für die Klassifikation wird daraus ein formatierter String erzeugt: `Title: Tweet by @...` und `Content:` Diese Darstellung ist bewusst einfach gehalten. Sie liefert dem Sprachmodell sowohl den eigentlichen Inhalt als auch einen minimalen

Kontext über die Autorschaft, ohne zusätzliche Komplexität einzuführen. In methodischer Hinsicht ist die Vorverarbeitung damit kein schweres Text-Cleaning, sondern eine zielgerichtete Normalisierung in eine LLM-freundliche Form.

4.2.3. Klassifikation

Die Klassifikation nutzt den in der Datei `tag-tree` abgelegten hierarchischen Tag-Baum als formale Zielstruktur. Dieser Baum wird zu Beginn des Programms eingelesen und anschließend an den Klassifikationsprompt übergeben. Die Funktion `classify_with_retry` in `src/classifiers.rs` übernimmt die Orchestrierung des Klassifikationsaufrufs. Intern ruft sie `classify` auf, welches über ein Subprozesskommando `codex` mit dem konstruierten Prompt startet. Damit ist das LLM nicht direkt als Bibliothek eingebunden, sondern über eine externe Prozessschnittstelle angebunden.

Die Modellantwort soll ausschließlich JSON enthalten. Dieses JSON wird in die Struktur `ClassificationResult` geparkt, die die Felder `tags`, `confidence`, `new_tags` und `reasoning` umfasst. Um mit unvollständigen Antworten umgehen zu können, sind die Felder mit `#[serde(default)]` versehen. Zusätzlich enthält die Funktion eine Retry-Logik: Scheitert entweder der Modellaufruf oder das Parsen der JSON-Antwort, wird der Vorgang bis zur maximalen Versuchsanzahl wiederholt. Diese Fehlerbehandlung ist für die Praxis wesentlich, weil freie Sprachmodelle zwar strukturierte Ausgaben erzeugen können, ihre Formatdisziplin aber nicht absolut garantiert ist.

4.2.4. Speicherung der Tags

Bereits beim Programmstart wird eine SQLite-Datenbank unter dem Pfad `resources.db` geöffnet und über `init_schema()` mit den benötigten Tabellen initialisiert. Das Datenbankschema umfasst die Tabellen `resources`, `tags`, `resource_tags` und `classification_log`. Dadurch werden sowohl die eigentlichen Ressourcen als auch ihre Zuordnungen, Konfidenzwerte und Begründungen der Klassifikation gespeichert. Die Wahl von SQLite ist für einen lokalen Prototyp zweckmäßig, weil keine separate Server-Infrastruktur nötig ist und relationale Tabellen dennoch zuverlässig angelegt werden können [10].

Vor einer neuen Verarbeitung prüft `resource_exists(url)`, ob die betreffende URL bereits bekannt ist. So kann verhindert werden, dass identische Ressourcen bei wiederholten Läufen erneut klassifiziert werden, sofern nicht der `--force`-Schalter gesetzt ist. Nach erfolgreicher Klassifikation wird die Ressource zunächst über `insert_resource` in die Tabelle `resources` eingetragen oder aktualisiert. Anschließend erzeugt `store_classification` die Tag-Zuordnungen in `resource_tags`, legt fehlende Hierarchieeinträge in `tags` an und schreibt die Begründung sowie Tag-Vorschläge in `classification_log`. Die Speicherung ist damit nicht nur Ergebnisablage, sondern strukturelle Grundlage für spätere Auswertung, Export und Weiterverarbeitung.

4.3. Implementierungsdetails

4.3.1. Projektstruktur

Das Projekt ist bewusst modular gehalten. Die zentrale Ablaufsteuerung liegt in `src/main.rs`, wo die Kommandozeilenschnittstelle mit `clap` definiert wird. Dort werden die drei Subkommandos `classify`, `export` und `stats` bereitgestellt. `classify` verarbeitet Ressourcen aus einer Eingabedatei, `export` schreibt gespeicherte Ressourcen in eine JSON-Datei und `stats` gibt eine einfache Übersicht über Ressourcen- und Tag-Anzahlen aus. Diese Aufteilung macht die Anwendung als Werkzeug bedienbar und trennt Steuerlogik von Fachlogik.

Die Klassifikationslogik selbst liegt in `src/classifiers.rs`, die Datenbankbindung in `src/db.rs` und die Scraper-spezifische Verarbeitung in `src/scrapers/twitter.rs`. Ergänzt wird diese Struktur durch externe Projektdateien wie `tag-tree`, `test-classification-list` und das Verzeichnis `scraped-tweets/`. Die Architektur folgt damit einer klaren Rollenverteilung: Eingabe und Ablaufsteuerung, Datenextraktion, semantische Klassifikation sowie Persistenz sind voneinander getrennt, aber über einfache Funktionschnittstellen gekoppelt.

4.3.2. Klassifikationslogik

Die interne Klassifikationslogik des Programms ist auf idempotente und nachvollziehbare Verarbeitung ausgelegt. Für jede URL wird zunächst geprüft, ob sie bereits in der Datenbank vorhanden ist. Existiert ein Eintrag und wurde kein `--force`-Flag gesetzt, wird die Ressource übersprungen. Damit bleibt der Lauf effizient und wiederholbar. Wird `--force` verwendet, wird eine bestehende Ressource erneut klassifiziert, was insbesondere bei verändertem Prompt oder erweitertem Tag-Baum sinnvoll ist.

Nach dem eigentlichen Modellaufruf werden die vom LLM gelieferten Informationen nicht blind übernommen, sondern zusätzlich ausgewertet. Das Programm gibt gefundene Tags, Konfidenzen und die textuelle Begründung auf der Konsole aus. Für neue Tag-Vorschläge wird der Elternknoten samt Begründung gesondert angezeigt. Außerdem wird mit `confident_tags(0.5)` eine einfache Schwelle angewandt, um sehr unsichere Klassifikationen sichtbar zu machen. Diese Schwelle ersetzt keine wissenschaftliche Evaluation, zeigt aber, wie modellinterne Sicherheitsschätzungen bereits im Prototyp in praktische Entscheidungslogik überführt werden können.

4.3.3. Scraper-Implementierung

Die Scraper-Implementierung ist ein gutes Beispiel für die pragmatische, prototypische Natur des Projekts. Anstatt eine vollständige Scraper-Logik direkt in Rust nachzubauen, nutzt das System ein bereits vorhandenes Python-Skript zur Extraktion von Tweet-Inhalten. Die Rust-Seite übernimmt die Rolle eines Wrappers: Sie extrahiert die Tweet-ID aus der URL, ruft das Skript mit dieser ID auf und prüft anschließend, ob die erwartete TOML-Datei erzeugt wurde. Dieser Ansatz beschleunigt die Entwicklung, weil vorhandene Werkzeuge wiederverwendet werden können.

Gleichzeitig zeigt die Implementierung, an welchen Stellen praktische Robustheit wichtig wird. Die durch den Scraper erzeugten TOML-Dateien folgen zwar einem erwarteten Grundschema, enthalten aber verschachtelte Felder und potenzielle Formatvarianten. Deshalb kombiniert der Parser eine reguläre `serde`-Deserialisierung mit einer manuellen Fallback-Auswertung einzelner Schlüssel wie `id`, `full_text` und `screen_name`. Für den Zweck des Prototyps ist dies ein sinnvoller Kompromiss zwischen Eleganz und Fehlertoleranz. Perspektivisch wäre denkbar, die Datenextraktion noch stärker zu standardisieren oder auf weitere Ressourcentypen auszudehnen. In der aktuellen Projektphase ist die bestehende Lösung jedoch ausreichend, um eine durchgängige Klassifikationspipeline realistisch zu demonstrieren.

Literaturverzeichnis

- [1] R. Liu und others, „Recent Advances in Hierarchical Multi-label Text Classification: A Survey“, *arXiv preprint arXiv:2307.16265*, 2023, [Online]. Verfügbar unter: <https://arxiv.org/abs/2307.16265>
- [2] C. D. Manning, P. Raghavan, und H. Schütze, *Introduction to Information Retrieval*. Cambridge University Press, 2008. doi: 10.1017/CBO9780511809071.
- [3] M. Guy und E. Tonkin, „Folksonomies: Tidying up Tags?“, *D-Lib Magazine*, Bd. 12, Nr. 1, 2006, [Online]. Verfügbar unter: <https://mirror.dlib.org/dlib/january06/guy/01guy.html>
- [4] G. Salton und C. Buckley, „Term-weighting approaches in automatic text retrieval“, *Information Processing & Management*, Bd. 24, Nr. 5, S. 513–523, 1988, doi: 10.1016/0306-4573(88)90021-0.
- [5] A. McCallum und K. Nigam, „A Comparison of Event Models for Naive Bayes Text Classification“, in *AAAI-98 Workshop on Learning for Text Categorization*, 1998, S. 41–48.
- [6] T. Joachims, *Learning to Classify Text Using Support Vector Machines*. Boston, MA: Springer, 2002. doi: 10.1007/978-1-4615-0907-3.
- [7] R. Bommasani und others, „On the Opportunities and Risks of Foundation Models“, *arXiv preprint arXiv:2108.07258*, 2021, [Online]. Verfügbar unter: <https://arxiv.org/abs/2108.07258>
- [8] A. Vaswani und others, „Attention Is All You Need“, in *Advances in Neural Information Processing Systems 30 (NeurIPS 2017)*, 2017. [Online]. Verfügbar unter: <https://arxiv.org/abs/1706.03762>

- [9] T. B. Brown und others, „Language Models are Few-Shot Learners“, *arXiv preprint arXiv:2005.14165*, 2020, [Online]. Verfügbar unter: <https://arxiv.org/abs/2005.14165>
- [10] SQLite Documentation, „CREATE TABLE“. [Online]. Verfügbar unter: https://www.sqlite.org/lang_createtable.html