

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	11
4.3.3. Scraper-Implementierung	12
5. Evaluation	12
5.1. Vergleich: Klassische Machine-Learning-Ansätze vs. Large Language Models	12
5.2. Analyse der Ergebnisse	12
5.3. Fehleranalyse	13
5.4. Stärken und Schwächen des Ansatzes	13
5.5. Ethische und gesellschaftliche Aspekte	14
6. Fazit	14
6.1. Zusammenfassung der Ergebnisse	14
6.2. Beantwortung der Forschungsfrage	15
6.3. Ausblick auf zukünftige Entwicklungen	15
Literaturverzeichnis	16

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 nach Liu et al. [1] entsteht.

Klassische Verfahren des Machine Learning erzielen nach Manning, Raghavan und Schütze [2] in klar umrissenen Anwendungsfällen zwar gute Ergebnisse, erfordern aber typischerweise eine vorgängige Merkmalsrepräsentation, Trainingsdaten und eine stabile Problemdefinition. 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, nach Manning, Raghavan und Schütze [2] aus einer größeren Informationsmenge diejenigen Elemente zu identifizieren, die für eine Anfrage oder einen Nutzungskontext relevant sind. 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 nach Guy und Tonkin [3] typische Probleme, etwa uneinheitliche Schreibweisen, Synonyme, Mehrdeutigkeiten und stark schwankende Detailgrade.

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 nach Manning, Raghavan und Schütze [2] aber stark davon ab, wie gut Texte vorverarbeitet, Merkmale ausgewählt und Trainingsdaten gelabelt wurden.

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 nach Manning, Raghavan und Schütze [2] konzeptionell einfach und für viele frühe Textklassifikationssysteme grundlegend gewesen.

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 [4] zeigen, dass unterschiedliche Gewichtungsschemata erheblichen Einfluss auf Retrieval- und Klassifikationsergebnisse haben können.

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 nach Manning, Raghavan und Schütze [2] wegen seiner Einfachheit und Effizienz weit verbreitet.

Für Textdaten existieren unterschiedliche Ereignismodelle, insbesondere multinomiale und Bernoulli-Varianten. McCallum und Nigam [5] zeigen, dass die Wahl des Ereignismodells die Klassifikationsleistung deutlich beeinflussen kann und dokumentieren die Eignung von Naive Bayes für Textklassifikation trotz der starken Unabhängigkeitsannahme. 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 nach Joachims [6] oft zuverlässig, weil viele Dokumente trotz großer Vokabularräume linear trennbar oder nahezu trennbar sind.

Für Textklassifikation ist die Methode deshalb attraktiv, weil sie mit TF-IDF- oder ähnlichen Repräsentationen gut kombinierbar ist. Joachims [6] zeigt, dass SVMs in diesem Bereich hohe Genauigkeit erreichen und insbesondere bei großen Merkmalsräumen robust arbeiten. 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 nach Manning, Raghavan und Schütze [2] 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.

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 nach Bommasani et al. [7] Modelle, die nicht nur lokale Wortfolgen, sondern auch komplexere sprachliche und semantische Zusammenhänge verarbeiten können.

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. [8] führen mit dem Transformer ein Modell ein, das auf Self-Attention basiert und dadurch Abhängigkeiten zwischen Wörtern eines Textes parallel und kontextsensitiv verarbeiten kann. 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 nach Bommasani et al. [7] auf Basis seines Trainings sehr viele sprachliche Muster, thematische Bezüge und typische Diskurszusammenhänge in seine Vorhersagen einfließen lassen. Für praktische Aufgaben wie Tagging ist genau diese Fähigkeit entscheidend, weil sie über rein oberflächenbasierte Schlüsselworterkennung 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. [9] 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. 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-Bericht 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 Projekt 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

Für die spätere Analyse des Prototyps 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.

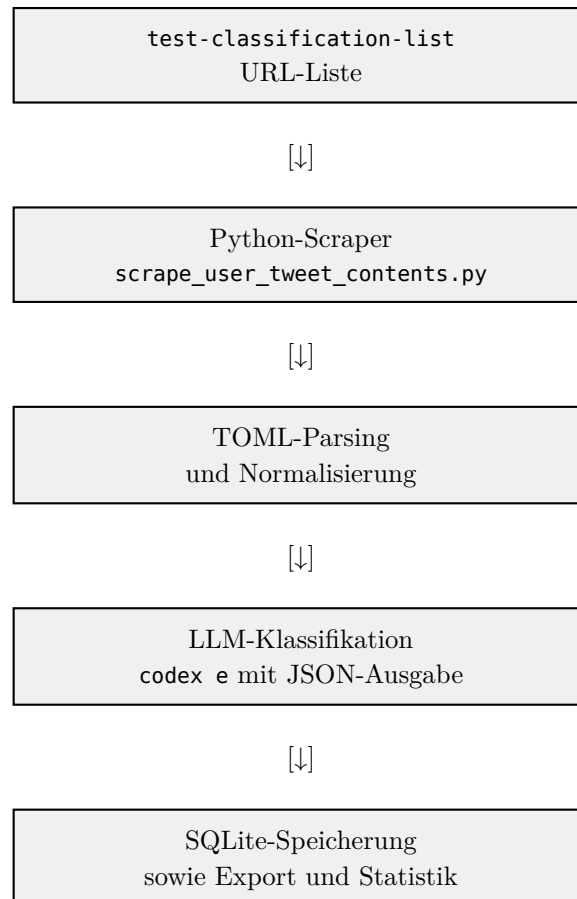
4.1.2. Eigenschaften und Herausforderungen der Daten

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`, wo die zentrale Funktion `classify_resources` alle Schritte der Pipeline koordiniert. Dort wird die Eingabedatei eingelesen, die Quelle einer URL grob bestimmt und anschließend nur dann weiterverarbeitet, wenn es sich um einen unterstützten Ressourcentyp handelt. Gegenwärtig betrifft das ausschließlich Links von X beziehungsweise Twitter. Diese Beschränkung ist bewusst gewählt, weil der Prototyp nicht möglichst viele Quellen gleichzeitig unterstützen soll, sondern zunächst einen zusammenhängenden, gut nachvollziehbaren Anwendungsfall.

Für unterstützte Links wird die Kernfunktion `scrapers::twitter::scrape(url)` aufgerufen. Sie übernimmt die Extraktion des eigentlichen Beitragsinhalts und lagert diesen Schritt an ein vorhandenes Python-Skript aus. Für die Gesamtarchitektur ist weniger die interne Hilfslogik entscheidend als die Rolle dieser Funktion innerhalb des Systems: Sie bildet die Schnittstelle zwischen der bloßen URL und einer lokal verfügbaren Rohrepräsentation des Beitrags. Damit verwandelt die Extraktion einen externen Verweis in eine Ressource, die durch die weitere Pipeline verarbeitet werden kann.

4.2.2. Vorverarbeitung

Nach der Extraktion wird die erzeugte Rohdatei in eine kompakte, einheitliche Textdarstellung überführt. Die dafür zentrale Funktion ist `parse_scraped_tweet`, die aus dem Scraper-Ergebnis genau jene Informationen herauszieht, die für die spätere Klassifikation tatsächlich benötigt werden: Beitragstext, Autor und Identifikationsdaten. Ziel dieser Phase ist nicht eine vollständige Rekonstruktion aller verfügbaren Metadaten, sondern eine sinnvolle Reduktion auf den inhaltlichen Kern der Ressource.

Das Ergebnis dieser Vorverarbeitung ist ein standardisiertes Eingabeformat für das Sprachmodell: **Title:** `Tweet by @...` und **Content:** `....`. Diese Normalisierung ist für das Gesamtsystem entscheidend, weil sie sehr kurze und formal uneinheitliche Social-Media-Texte in eine konsistente Repräsentation überführt. Dadurch kann die Klassifikation später mit einer gleichbleibenden Struktur arbeiten, auch wenn die ursprünglichen Beiträge stark variieren.

4.2.3. Klassifikation

Die Klassifikation bildet das inhaltliche Zentrum des gesamten Prototyps. Grundlage ist der in **tag-tree** abgelegte hierarchische Tag-Baum, der dem Sprachmodell als Zielstruktur vorgegeben wird. Die zentrale Funktion dafür ist `classify_with_retry` in `src/classifiers.rs`. Sie bündelt den eigentlichen Modellaufruf, die Prüfung der Rückgabe und die Wiederholung des Vorgangs, falls das Ergebnis nicht im erwarteten Format vorliegt. Damit ist sie nicht nur eine technische Hilfsfunktion, sondern der Punkt, an dem semantische Zuordnung und Systemrobustheit zusammenkommen.

Inhaltlich besteht die Aufgabe der Klassifikation darin, aus einer knappen Ressourcendarstellung eine kleine Menge möglichst spezifischer Tags abzuleiten. Im aktuellen Stand dient die Codex-CLI dabei als Harness; aufgerufen wird `codex e --model gpt-5.4-mini`. Die Antwort des Modells wird als JSON mit den Feldern `tags`, `confidence`, `new_tags` und `reasoning` verarbeitet. Für das Gesamtverständnis des Systems ist daher vor allem wichtig, dass die Klassifikation nicht als freier Text endet, sondern in eine strukturierte Ausgabe überführt wird, die direkt weiterverarbeitet werden kann. So entsteht die Brücke zwischen sprachlicher Interpretation und maschineller Speicherung.

4.2.4. Speicherung der Tags

Die Speicherung ist der Schritt, in dem aus einer vorübergehenden Modellantwort ein dauerhaft nutzbares Ergebnis wird. Die zentrale Funktion hierfür ist `store_classification` in `src/db.rs`. Sie übernimmt die vom Sprachmodell erzeugten Tags, Konfidenzen und Begründungen und überführt sie in die relationale Struktur der SQLite-Datenbank. Auf diese Weise werden Ressourcen, Tag-Zuordnungen und Klassifikationsbegründungen nicht nur einmalig angezeigt, sondern für spätere Suchen, Exporte und Auswertungen konserviert.

Für das große Ganze ist vor allem wichtig, dass die Datenbank mehrere Rollen gleichzeitig erfüllt. Sie speichert nicht nur den Inhalt einer Ressource, sondern auch ihre thematische Einordnung und die Begründung dieser Einordnung. Vorschläge für neue Tags werden zusätzlich in einer eigenen Warteschlange gespeichert. Erreicht eine Klassifikation eine Mindestkonfidenz von **0.75**, kann ein vorgeschlagener neuer Tag automatisch in die Datei **tag-tree** übernommen werden. Liegt die Konfidenz darunter, bleibt der Vorschlag für eine manuelle Prüfung über den Befehl `classifier review-tags` erhalten. Die Speicherung bildet damit den Abschluss der Pipeline und zugleich den Ausgangspunkt für Statistik, Export und Taxonomiepflege.

4.3. Implementierungsdetails

4.3.1. Projektstruktur

Die Projektstruktur folgt einer bewusst einfachen Arbeitsteilung. `src/main.rs` übernimmt die zentrale Steuerung der Anwendung, `src/classifiers.rs` bündelt die semantische Klassifikation, `src/scrapers/twitter.rs` ist für die Gewinnung der Tweet-Inhalte zuständig und `src/db.rs` organisiert die dauerhafte Speicherung. Ergänzt wird diese Struktur durch die externen Dateien **tag-tree** als Wissensstruktur und **test-classification-list** als Eingabe. Auf diese Weise bleibt die Architektur überschaubar und zugleich klar genug gegliedert, um einzelne Teile unabhängig voneinander weiterentwickeln zu können.

Für die Facharbeit ist dabei weniger die vollständige Auflistung aller Module wichtig als die Grundidee der Architektur: Das System trennt Eingabe, Extraktion, Interpretation und Speicherung in eigenständige Verantwortungsbereiche. Diese Trennung erleichtert nicht nur die technische Wartung, sondern macht auch die konzeptionelle Logik des Projekts sichtbar.

4.3.2. Klassifikationslogik

Die Klassifikationslogik des Projekts lässt sich als Zusammenspiel weniger zentraler Entscheidungen beschreiben. Erstens soll eine Ressource nicht einfach irgendeinen, sondern einen möglichst spezifischen Tag erhalten. Zweitens darf eine Ressource mehreren Themenbereichen zugeordnet werden. Drittens muss das System mit Unsicherheit umgehen können, statt jede Eingabe mit derselben Entschiedenheit zu behandeln. Diese drei Anforderungen prägen sowohl den Prompt als auch die weitere Verarbeitung der Ergebnisse.

Im großen Ganzen bedeutet das: Das Sprachmodell dient nicht nur als Textgenerator, sondern als semantischer Einordner innerhalb einer vorgegebenen Wissensstruktur. Die resultierenden Tags, Begründungen und Unsicherheiten werden anschließend so verarbeitet, dass sie für den Nutzer nachvollziehbar bleiben. Genau in dieser Verbindung aus inhaltlicher Interpretation und technischer Struktur liegt der eigentliche Kern der Klassifikationslogik.

4.3.3. Scraper-Implementierung

Die Scraper-Implementierung verdeutlicht den prototypischen Charakter des Projekts. Anstatt sämtliche Schritte selbst neu zu entwickeln, nutzt das System für die Extraktion ein vorhandenes Python-Werkzeug und bindet es in die Rust-Anwendung ein. Aus Sicht des Gesamtsystems ist dies sinnvoll, weil die Extraktion nicht Selbstzweck ist, sondern vor allem die Voraussetzung dafür schafft, dass aus einer URL überhaupt ein klassifizierbarer Text wird.

Entscheidend ist daher weniger, wie jede einzelne Hilfsroutine des Scrapers intern arbeitet, sondern welche Funktion der Scraper im Gesamtprozess erfüllt: Er stellt sicher, dass flüchtige Plattforminhalte in eine lokal verfügbare und weiterverarbeitbare Form überführt werden. Erst dadurch können die nachgelagerten Schritte der Normalisierung, Klassifikation und Speicherung zuverlässig anschließen.

5. Evaluation

5.1. Vergleich: Klassische Machine-Learning-Ansätze vs. Large Language Models

Der Vergleich zwischen klassischen Verfahren des Machine Learning und Large Language Models zeigt, dass beide Ansätze unterschiedliche Stärken besitzen und deshalb nicht pauschal gegeneinander ausgespielt werden sollten. Klassische Verfahren wie Naive Bayes oder Support Vector Machines sind besonders dann sinnvoll, wenn ein klar definierter Anwendungsfall, ein stabiles Label-Set und ausreichend annotierte Trainingsdaten vorliegen. Unter diesen Bedingungen lassen sich nach Manning, Raghavan und Schütze [2], McCallum und Nigam [5] sowie Joachims [6] reproduzierbare Modelle trainieren, die effizient arbeiten und deren Verhalten in Bezug auf Eingabemerkmale oft leichter analysierbar ist.

Für das vorliegende Projekt lagen diese Voraussetzungen jedoch gerade nicht vor. Es existierte zwar ein Tag-Baum, aber kein ausreichend großes, sauber gelabeltes Korpus. Zudem bestand die Aufgabe nicht in der Vorhersage einer einzigen festen Klasse, sondern in der Auswahl mehrerer möglichst spezifischer Tags innerhalb einer Hierarchie. Hinzu kommt, dass die verarbeiteten Texte kurz, kontextabhängig und teilweise implizit formuliert sind. In einem solchen Szenario bieten Large Language Models nach Brown et al. [9] und Bommasani et al. [7] einen praktischen Vorteil, weil sie ohne projektspezifisches Training semantische Beziehungen nutzen und durch Prompting auf neue Aufgaben ausgerichtet werden können.

Gleichzeitig machen die Beobachtungen aus dem Prototyp deutlich, dass diese Flexibilität mit Kosten verbunden ist. LLMs liefern keine vollständig deterministischen Ergebnisse, ihre Konfidenzwerte sind nicht statistisch kalibriert, und die Qualität der Ausgabe hängt stark von Prompt, Modellzustand und vorhandener Tag-Struktur ab. Klassische Verfahren wären in einer späteren Projektphase dann wieder attraktiv, wenn ein größerer, manuell überprüfter Goldstandard zur Verfügung stünde. Für den prototypischen Einstieg ist der LLM-Ansatz jedoch nachvollziehbar überlegen, weil er mit geringem Vorbereitungsaufwand eine inhaltlich brauchbare Erstklassifikation ermöglicht.

5.2. Analyse der Ergebnisse

Für die eigentliche Evaluation wurde der Prototyp auf sechs neuen X-Beiträgen aus dem Themenfeld Compilerbau und Programmiersprachen ausgeführt. Alle sechs Ressourcen erhielten mindestens einen gespeicherten Tag. Insgesamt wurden elf Tag-Zuordnungen gespeichert. Der Mittelwert der gespeicherten Konfidenzen lag bei rund **0.91**; der niedrigste Wert lag bei **0.80**, der höchste bei **0.99**. Wegen der kleinen Stichprobe sind diese Werte nicht als allgemeingültige Leistungsmetriken zu verstehen, sie zeigen aber deutlich, dass die Pipeline auf diesem Datensatz stabil arbeitet und fachlich spezifische Ergebnisse erzeugen kann.

Besonders überzeugend ist die inhaltliche Präzision der vergebenen Tags. Ein Tweet über eine Einführung in Compiler wurde mit `cs/theory/compiler` klassifiziert. Ein anderer Beitrag mit Ressourcen zu Registerallokation, SSA, Codegenerierung und Optimierung wurde passend unter `cs/theory/compiler/code_generation`, `cs/theory/compiler/optimization` und `cs/theory/compiler/analysis` einsortiert. Für einen Beitrag über den Optimizer-Workflow von LLVM vergab das System `cs/theory/compiler/llvm` und `cs/theory/compiler/optimization`. Die Empfehlung des Buchs *Types and Programming Languages* wurde mit `cs/theory/compiler/type_systems` und `cs/theory/type_theory` klassifiziert. Auch der Zig-Beitrag wurde sinnvoll erfasst: Er erhielt `cs/theory/compiler/parsing` und `cs/programming_languages/zig`.

Diese Ergebnisse sprechen für eine hohe Effektivität des Ansatzes in inhaltlich klaren Fachfällen. Das Modell bleibt nicht auf grobe Oberbegriffe beschränkt, sondern nutzt die Hierarchie tatsächlich bis auf spezifische Blätter wie `llvm`, `code_generation`, `optimization`, `history`, `type_systems` oder `type_theory`. Zugleich zeigt der Datensatz, dass das System nicht nur vorhandene Kategorien nutzt, sondern die Taxonomie bei Bedarf sinnvoll erweitert. Beim Zig- und Lexer-Beitrag schlug das Modell `cs/theory/compiler/lexical_analysis` vor. Da die zugehörige Mindestkonfidenz bei 0.80 lag und damit oberhalb des festgelegten Schwellenwerts, wurde dieser Tag automatisch in den Tag-Baum übernommen. Gerade dieser Fall zeigt, dass das Verfahren nicht nur klassifiziert, sondern die bestehende Struktur auch kontrolliert verfeinern kann.

5.3. Fehleranalyse

Die Fehleranalyse zeigt im aktuellen Stand weniger grobe semantische Fehlklassifikationen als vielmehr Grenzen der Taxonomie und der Modellkonsistenz. So wurde der Beitrag über *Types and Programming Languages* nicht nur unter `cs/theory/type_theory`, sondern zusätzlich unter `cs/theory/compiler/type_systems` eingeordnet. Diese Entscheidung ist inhaltlich noch vertretbar, zeigt aber, dass die Grenzen zwischen benachbarten Unterbereichen der Hierarchie nicht immer trennscharf gezogen werden. Ähnliche Überschneidungen können auch bei Tags wie `analysis`, `optimization` oder `history` auftreten, wenn ein kurzer Beitrag mehrere Aspekte zugleich berührt.

Auf technischer Ebene traten während der Entwicklung ebenfalls zwei relevante Probleme auf. Erstens zeigte sich, dass eine erzwungene Neuklassifikation mit `--force` zunächst alte und neue Tag-Zuordnungen kumulierte, anstatt die bestehende Zuordnung eines Beitrags zu ersetzen. Zweitens erwies sich eine nachträgliche Normalisierung von Tag-Pfaden als zu unflexibel, weil sie fehlerhafte Modellantworten verdecken konnte. Beide Punkte wurden behoben, verdeutlichen aber, dass die Zuverlässigkeit des Gesamtsystems nicht allein vom Modell abhängt, sondern ebenso von der sauberen Nachverarbeitung seiner Ausgabe.

Die neue Taxonomie-Workflow-Regelung bringt zudem einen bewussten Zielkonflikt mit sich. Automatische Übernahmen oberhalb von 0.75 beschleunigen die Weiterentwicklung des Tag-Baums, beruhen aber weiterhin auf heuristischen Konfidenzwerten des Modells. Deshalb bleibt die manuelle Prüfung über `classifier review-tags` für schwächere oder zweifelhafte Vorschläge notwendig. Die Fehleranalyse zeigt damit, dass nicht nur semantische Treffsicherheit, sondern auch ein klug gestalteter Kontrollmechanismus zur Qualität des Systems beiträgt.

5.4. Stärken und Schwächen des Ansatzes

Eine wesentliche Stärke des entwickelten Ansatzes liegt in seiner unmittelbaren praktischen Einsetzbarkeit ohne vorgängige Trainingsphase. Der aktuelle Testlauf zeigt, dass bereits mit einem bestehenden Tag-Baum und ohne gelabeltes Korpus fachlich differenzierte Zuordnungen entstehen können. Gerade bei kurzen technischen Beiträgen ist das ein relevanter Vorteil, weil hier die Kombination aus Weltwissen, Kontextverstehen und hierarchischer Einordnung besonders gefragt ist. Die hohe Spezifität der vergebenen Tags im Compiler-Datensatz spricht klar dafür, dass der LLM-Ansatz für solche Ressourcen sehr effektiv ist.

Eine weitere Stärke ist die Verbindung von Klassifikation und Taxonomiepflege. Der Prototyp speichert nicht nur Tags und Begründungen, sondern kann neue Kategorien kontrolliert in die bestehende Struktur übernehmen. Der automatisch ergänzte Tag `cs/theory/compiler/lexical_analysis` zeigt, dass das

System Lücken der Hierarchie nicht nur erkennt, sondern in klaren Fällen auch praktisch schließen kann. Gleichzeitig bleibt durch die Review-Funktion eine menschliche Kontrollinstanz für unsichere Vorschläge erhalten. Hinzu kommt die saubere Modularisierung in Extraktion, Vorverarbeitung, Klassifikation und Speicherung, die das System technisch überschaubar und erweiterbar macht.

Den Stärken stehen dennoch mehrere Schwächen gegenüber. Erstens sind die modellseitigen Konfidenzwerte heuristisch und nicht im statistischen Sinn kalibriert. Zweitens hängt die Qualität stark von der vorhandenen Taxonomie ab. Fehlen passende Kategorien, kann das Modell zwar neue Tags vorschlagen, doch deren automatische Übernahme bleibt eine Abwägung zwischen Tempo und Kontrolle. Drittens besteht eine Abhängigkeit von externen Komponenten, insbesondere vom Python-Scraper und vom LLM-Aufruf über die Codex-CLI. Viertens ist die empirische Basis weiterhin klein und thematisch eng auf Compiler- und Programmierspracheninhalte konzentriert. Der Ansatz ist daher als Prototyp bereits überzeugend, benötigt für eine allgemeine Bewertung jedoch breitere Daten und systematischere Vergleichsmaßstäbe.

5.5. Ethische und gesellschaftliche Aspekte

Die Verwendung von LLMs zur automatisierten Ordnung persönlicher oder öffentlicher Informationssammlungen wirft auch ethische und gesellschaftliche Fragen auf. Zunächst betrifft dies die Verzerrungen der Modelle selbst. Sprachmodelle übernehmen nach Bommasani et al. [7] sowie Bender et al. [10] aus ihren Trainingsdaten implizite Muster, Prioritäten und Vorurteile. Diese können sich nicht nur in generierten Texten, sondern auch in Klassifikationsentscheidungen niederschlagen. Wenn ein Modell bestimmte Themengebiete systematisch anders gewichtet, beeinflusst es damit auch, wie Wissen im Zielsystem strukturiert und wiedergefunden wird.

Ein zweiter Aspekt betrifft Transparenz und Nachvollziehbarkeit. Für Nutzerinnen und Nutzer ist es oft nicht ohne Weiteres ersichtlich, warum ein bestimmter Tag vergeben oder ein anderer verworfen wurde. Das Projekt reagiert darauf teilweise mit dem Feld `reasoning`, doch auch diese textliche Begründung bleibt letztlich modellgeneriert und garantiert keine echte Erklärbarkeit. In produktiven Anwendungen wäre daher zu überlegen, wie automatische Vorschläge mit menschlicher Überprüfung kombiniert werden können, damit die Organisation von Wissen nicht vollständig in eine schwer kontrollierbare Black Box ausgelagert wird.

Schließlich sind auch Daten- und Plattformfragen relevant. Der aktuelle Prototyp verarbeitet Inhalte von X beziehungsweise Twitter und ist damit von einer externen Plattform abhängig, deren Datenzugang, Formate und Nutzungsbedingungen sich ändern können. Außerdem stellt sich die Frage, welche Inhalte automatisiert gespeichert, analysiert und weiterverarbeitet werden dürfen. Selbst wenn die Anwendung im privaten Kontext bleibt, zeigt sie exemplarisch ein breiteres gesellschaftliches Spannungsfeld: Automatisierte Wissensorganisation kann Produktivität erhöhen, verschiebt aber zugleich Verantwortung von Menschen auf Modelle, deren Entscheidungen nicht neutral sind. Eine reflektierte Nutzung solcher Systeme setzt deshalb nicht nur technische, sondern auch gesellschaftliche Urteilskraft voraus.

6. Fazit

6.1. Zusammenfassung der Ergebnisse

Die Arbeit hat gezeigt, dass die automatisierte Verschlagwortung digitaler Ressourcen mit Large Language Models sowohl theoretisch begründbar als auch praktisch umsetzbar ist. Ausgehend vom Problem einer wachsenden, schwer überschaubaren Informationsmenge wurde zunächst deutlich, dass klassische Verfahren des Machine Learning in diesem Anwendungsfall zwar wichtige Grundlagen liefern, aber ohne annotierte Trainingsdaten und bei kurzen, kontextreichen Texten an Grenzen stoßen. Darauf aufbauend wurde ein LLM-basierter Ansatz entwickelt, der die Klassifikation direkt über einen strukturierten Prompt vornimmt und die Ergebnisse in einer relationalen Datenbank speichert.

Mit dem implementierten Prototyp liegt nun eine durchgängige Pipeline vor, die URLs aus einer Eingabedatei einliest, X-Beiträge extrahiert, deren Inhalte normalisiert, eine hierarchische Multi-Label-Klassifikation durchführt und die Resultate speichert. Im konkreten Evaluationslauf mit sechs neuen

Ressourcen konnten alle sechs Beiträge direkt und mit insgesamt elf fachlich plausiblen Tag-Zuordnungen klassifiziert werden. Die Ergebnisse deckten zentrale Unterbereiche wie Compiler-Optimierung, Codegenerierung, LLVM, Parsing, Zig, Type Systems und Type Theory ab. Zusätzlich wurde mit `cs/theory/compilers/lexical_analysis` ein neuer, modellseitig vorgeschlagener Tag automatisch in den Baum aufgenommen. Damit zeigt der Prototyp nicht nur Klassifikationsfähigkeit, sondern auch eine erste Form kontrollierter struktureller Weiterentwicklung.

6.2. Beantwortung der Forschungsfrage

Die leitende Forschungsfrage lautete, inwieweit sich Large Language Models dazu eignen, kurze digitale Ressourcen ohne spezielles domänenspezifisches Training automatisch in ein hierarchisches Multi-Label-Tagging-System einzuordnen. Auf Basis der theoretischen Einordnung und der prototypischen Umsetzung lässt sich diese Frage klar positiv beantworten. Der durchgeführte Testlauf zeigt, dass LLMs kurze technische Ressourcen bereits ohne projektspezifisches Training sehr wirksam semantisch einordnen können, sofern der Tag-Baum passende fachliche Ankerpunkte bereitstellt. Gerade bei kontextreichen Kurztexten aus dem Bereich Compilerbau und Programmiersprachen erwies sich der Ansatz als erstaunlich treffsicher und ausreichend spezifisch.

Ihre Eignung ist jedoch nicht grenzenlos. Die Qualität der Ergebnisse hängt stark von der Güte des Prompts, der Vollständigkeit der Tag-Hierarchie und der Robustheit der technischen Pipeline ab. Außerdem sinkt die Zuverlässigkeit bei inhaltsarmen oder stark dialogischen Beiträgen deutlich. LLMs ersetzen daher keine sorgfältige Modellierung des Zielsystems, sondern verschieben den Schwerpunkt von der Trainingsphase hin zur Prompt-Gestaltung, Ergebnisvalidierung und Taxonomiepflege. Für das hier untersuchte Projekt bedeutet das: Das Sprachmodell ist kein perfekter automatischer Entscheider, aber bereits jetzt ein sehr leistungsfähiges Werkzeug zur semantischen Vorstrukturierung von Ressourcen.

6.3. Ausblick auf zukünftige Entwicklungen

Für die Weiterentwicklung des Projekts ergeben sich mehrere sinnvolle Schritte. Erstens sollte die Datenbasis deutlich erweitert werden, um die Qualität der Klassifikation auf einer breiteren Grundlage beurteilen zu können. Zweitens wäre eine manuelle Referenzannotation einzelner Ressourcen hilfreich, um systematischer zwischen gelungenen und fehlerhaften Zuordnungen unterscheiden zu können. Drittens sollte die Statusverwaltung der Pipeline verbessert werden, damit fehlgeschlagene Klassifikationen explizit erkannt, erneut angestoßen oder zur manuellen Prüfung markiert werden können.

Darüber hinaus sollte der bereits eingeführte Prüfprozess für neue Tag-Vorschläge weiter ausgebaut werden, etwa durch feinere Schwellenwerte, Sammelansichten oder explizite Begründungsbewertungen. Ebenso naheliegend ist eine Ausweitung auf weitere Ressourcentypen wie klassische Bookmarks, Blogartikel, Videos oder wissenschaftliche Publikationen. Langfristig könnte aus dem aktuellen Prototyp so ein System entstehen, das nicht nur persönliche Informationssammlungen organisiert, sondern auch als Werkzeug für individuelle Wissensarbeit, Recherche und digitale Archivierung dient.

Literaturverzeichnis

- [1] R. Liu und others, „Recent Advances in Hierarchical Multi-label Text Classification: A Survey“, *arXiv preprint arXiv:2307.16265*, 2023.
- [2] C. D. Manning, P. Raghavan, und H. Schütze, *Introduction to Information Retrieval*. Cambridge University Press, 2008.
- [3] M. Guy und E. Tonkin, „Folksonomies: Tidying up Tags?“, *D-Lib Magazine*, Bd. 12, Nr. 1, 2006.
- [4] G. Salton und C. Buckley, „Term-weighting approaches in automatic text retrieval“, *Information Processing & Management*, Bd. 24, Nr. 5, S. 513–523, 1988.
- [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.
- [7] R. Bommasani und others, „On the Opportunities and Risks of Foundation Models“, *arXiv preprint arXiv:2108.07258*, 2021.
- [8] A. Vaswani und others, „Attention Is All You Need“, in *Advances in Neural Information Processing Systems 30 (NeurIPS 2017)*, 2017.
- [9] T. B. Brown und others, „Language Models are Few-Shot Learners“, *arXiv preprint arXiv:2005.14165*, 2020.
- [10] E. M. Bender, T. Gebru, A. McMillan-Major, und S. Shmitchell, „On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?“, in *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, 2021, S. 610–623.