FernUniversität in Hagen
Fakultät für Mathematik und Informatik

AIG
Artificial Intelligence Group

Auf maschinellem Lernen gestützte Zuordnung von Bedienelementen für Testanweisungen in sicherheitskritischen Lokomotiv-Tests

Masterarbeit

zur Erlangung des Grades *Master of Science* (*M.Sc.*)
im Studiengang Praktische Informatik

vorgelegt von
Wiebke Eva Albers

Erstgutachter: Prof. Dr. Matthias Thimm
Artificial Intelligence Group, Fernuniversität Hagen

Betreuer: Dr.-Ing. Manuel Gesell
Alstom Transportation Germany GmbH

Erklärung

Ich erkläre, dass ich die Masterarbeit selbstständig und ohne unzulässige Inanspruchnahme Dritter verfasst habe. Ich habe dabei nur die angegebenen Quellen und Hilfsmittel verwendet und die aus diesen wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht. Die Versicherung selbstständiger Arbeit gilt auch für enthaltene Zeichnungen, Skizzen oder graphische Darstellungen. Die Masterarbeit wurde bisher in gleicher oder ähnlicher Form weder derselben noch einer anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht. Mit der Abgabe der elektronischen Fassung der endgültigen Version der Masterarbeit nehme ich zur Kenntnis, dass diese mit Hilfe eines Plagiatserkennungsdienstes auf enthaltene Plagiate geprüft werden kann und ausschließlich für Prüfungszwecke gespeichert wird.

Der Veröffentlichung dieser Arbeit auf der Webseite des Lehrgebiets Künstliche Intelligenz und damit dem freien Zugang zu dieser Arbeit stimme ich ausdrücklich zu.

Für diese Arbeit erstellte Software wurde quelloffen verfügbar gemacht, ein entsprechender Link zu den Quellen ist in dieser Arbeit enthalten. Gleiches gilt für angefallene Forschungsdaten.

Mannheim, 08.04.2026 .....

(Ort, Datum) (Unterschrift)

Zusammenfassung

Die Durchführung sicherheitsrelevanter Systemtests bei Lokomotiven ist bislang stark manuell geprägt, was zeitintensiv und fehleranfällig ist. Zwar existiert bereits ein Prototyp zur teilautomatisierten Ausführung solcher Tests, doch die Transformation natürlicher Testanweisungen in ausführbare Testschritte, voran deren Zuordnung zu den zugehörigen Bedienelementen, stellt weiterhin eine zentrale Herausforderung dar. Ziel dieser Arbeit ist die Entwicklung eines intelligenten Systems auf Basis maschinellen Lernens (ML), welches diesen Transformationsprozess effizient und ressourcenschonend unterstützt. Der Fokus liegt dabei auf der Eignung eines klein-gewichtigen Transformer-Modells zur semantischen Umsetzung der Testanweisungen und einer anschließenden Zuordnung von Testschritten zu Bedienelementen unter Verwendung von semantischer Textähnlichkeit und eingeschränkter Ressourcenverfügbarkeit.

Abstract

The execution of safety critical system tests on locomotives is currently characterized by a high degree of manual effort, resulting in a time-consuming and error-prone process. Although a prototype for the semi-automated execution of these tests is already available, the transfromation of natural language test instructions into executable test steps, above all the assignment to corresponding control elements, remains a core challenge. This work aims to develop an intelligent system based on machine learning (ML) to efficiently support this transformation process. The focus is placed on evaluating the suitability of a light-weight transformer model for the semantic transformation of test instructions and a subsequent mapping of test steps to corresponding control elements using semantic text similarity respecting limited resource availability.

Inhaltsverzeichnis

1 Einleitung

Im Zuge der folgenden Einleitung wird die Basis dieser Arbeit vorgestellt. Diese beinhaltet eine Beschreibung der kontextuellen Hintergründe und eine ausführliche Darstellung der Problemstellung (engl. Thesis Statement (TS)). Diese resultiert aus der Motivation und bildet die Grundlage für die zu behandelnden Forschungsfragen (engl. Research Questions (RQ)). Die RQs stellen die Ziele dar, die innerhalb der Arbeit behandelt und beantwortet werden sollen. Zudem wird die Methodik vorgestellt, die für das Erreichen der Ziele verwendet wird und die Struktur der Arbeit maßgeblich mitgestaltet.

1.1 Motivation

Lokomotiven sind ein unverzichtbarer Bestandteil im öffentlichen Güter- und Personentransport und zählen zu den sichersten Transportmitteln in Europa [4]. Um die funktionale Sicherheit (FuSi) im operativen Betrieb zu gewährleisten, sind während des Entwicklungsprozesses zahlreiche Tests nötig. Neben bereits automatisierten Tests der logischen Systemfunktionen werden Validierungstests durchgeführt, um die Funktionsfähigkeit sicherheitskritischer Komponenten im Kontext realer Zughardware nachzuweisen. Ein Teil der Tests erfordern händische Eingaben und Hardwarebetätigung durch den Tester, was eine manuelle Ausführung erfordert und einen erheblichen Zeitaufwand und auch Fehleranfälligkeit bedeutet. Zudem ist ein fundiertes Wissen um die Bedienung einer Lokomotive nötig, was wiederum eine umfangreiche Schulung des Personals erfordert. Die manuelle Interpretation und Ausführung der vielen Testfälle ist ein Prozess, der sich durch eine Teilautomatisierung erheblich beschleunigen lässt. Der Prototyp eines solchen Systems wurde in den letzten Jahren in der Firma Alstom bereits entwickelt. Es ist in der Lage, funktionale Testspezifikationen (FTS, engl. functional test specifications) in sequenzielle Testschritte umzuwandeln, welche mit dem zugehörigen Programm NI TestStand [29] an einem Prüfturm automatisiert ausgeführt werden. Die Transformation der Testanweisungen in zugehörige Testschritte unterliegt zahlreichen Herausforderungen, die sich durch die zielgerichtete Unterstützung eines intelligenten Systems effizient bewältigen lassen. Eine dieser Herausforderungen besteht in der Identifikation von Bedienelementen im Betriebsmittelverzeichnis (BMV), über welche zugehörige Signale für die Simulationssoftware angesprochen werden. Die Entwicklung eines solchen intelligenten Systems auf Basis maschinellen Lernens (ML) zur Identifikation der Bedienelemente ist Ziel dieser Arbeit.

1.2 Problemstellung

Im Zuge des Entwicklungsprozesses der Zugsoftware werden zahlreiche Tests durchlaufen. Diese umfassen klassische Software-Tests und Validierungstests. Klassische Software-Tests prüfen lediglich das logische Verhalten einzelner Funktionen oder Systeme. Dabei ist es ausreichend, den Input des Systems in der Software zu forcieren und den Output mit dem erwarteten Ergebnis zu vergleichen. Da es sich dabei um rein softwarebasierte Tests handelt, ist eine Automatisierung einfach umsetzbar und wird bei Alstom bereits im Software-Testing eingesetzt.

Die Validierungstests der FuSi prüfen im Gegensatz dazu das Gesamtsystem, was reale Hardwareinteraktionen einschließt. Die Tests lassen sich somit nicht mehr ausschließlich durch das Forcieren von Signalen durchführen, was die Automatisierbarkeit einschränkt und auch die Hauptproblematik des Themas darstellt. Ein bestehender Prototyp ermöglicht bereits die teilautomatisierte Ausführung der Validierungstests mithilfe der Testmanagementsoftware NI TestStand. Eine zentrale Herausforderung liegt in der semantischen Zuordnung von Testanweisungen zu deren zugehörigen Hardware-Systemkomponenten im Betriebsmittelverzeichnis. Dies schließt insbesondere eindeutige Identifikatoren für Bedienelemente im Stromlaufplan ein, die sogenannten E3-Namen. Dies sind vom Tool E3.series [12] (CAE-Programm zur Erstellung von Stromlaufplänen) vergebene Bezeichner, welches jedes elektrisches Bauteil eindeutig adressiert und auf ein passendes Schnittstellensignal der Software gemappt werden kann. Dieser Bestandteil der Testautomatisierung stellt als Mehrklassen-Klassifizierung einen zentralen Teil der Teilautomatisierung dar und bildet somit die Basis für die folgende Erstellung der Testsequenz.

Im bestehenden Prototyp erfolgt die Zuordnung aktuell über einen heuristischen Stringvergleich, welcher bei komplexen oder variierenden Formulierungen schnell an seine Grenzen stößt. Das Hauptproblem ist hierbei, dass semantisch ähnliche Formulierungen durch andere Satzstrukturen oder Wortwahl nicht als ähnlich erkannt werden. So besitzen die folgenden beispielhaften Formulierungen den selben inhaltlichen Sinn von „fahren“: „Traktion geben“, „TBC-Hebel auf Position T“, „v>0 km/h“. Hinzu kommen Variationen in der Wortreihenfolge, welche die syntaktische Vielfältigkeit erhöhen.

Alternativen zum Stringvergleich wie keyword-basierte Erkennung unterliegen ähnlichen Herausforderungen. Durch nicht-standardisiertes Vokabular besitzen Schlüsselwörter Mehrdeutigkeiten, welche sich erst im Kontext des Satzes auflösen. Ein Beispiel ist „Traktion geben“ (bedeutet fahren/beschleunigen des Zuges), „Multi-Traktion“ (beschreibt die Kopplung mehrerer Lokomotiven, wobei eine Lok die Steuerung der anderen übernimmt) oder „Traktionssperre“ (beschreibt den Zustand, in dem das Beschleunigen der Lokomotive aktiv verhindert wird). Das Schlüsselwort „Traktion“ kommt in jeder Formulierung vor, repräsentiert jedoch

jedes mal einen anderen Kontext. So müssten für jedes Betriebsmittel umfangreiche Regeln für die Schlüsselwörter erstellt werden, welche jegliche Synonyme, Mehrdeutigkeiten und Wortverwandtschaften einschließen. Bei mehreren Übereinstimmungen wird zudem nur der erste Treffer verwendet und keine weitere Unterscheidung vorgenommen. Abgesehen von der Effizienz der Zuordnung besteht bei Schlüsselwort-basierter Erkennung ein fortlaufender Pflegeaufwand der entsprechenden Datenbank. Die Testspezifikationen verändern sich regelmäßig durch sich ändernde Anforderungen, was auch eine analoge Anpassung der Schlüsselwörter zur Folge hätte.

Die Integration eines intelligenten Systems auf Basis maschinellen Lernens verspricht an dieser Stelle effizientere Ergebnisse. Hier handelt es sich um eine Analyse auf semantischer Ebene, wodurch Synonyme, Tippfehler und Umschreibungen der selben Intention trotz der Verwendung unterschiedlicher Wörter als ähnlich erkannt werden. Durch die Möglichkeit, dem Modell im laufenden Betrieb Feedback zu seinen Prognosen zu liefern, werden diese fortlaufend verbessert. Neue Formulierungen lassen sich schneller einbinden und Fehler korrigieren. Zudem reduziert sich der nachträgliche Wartungsaufwand, da sich falsche Vorhersagen unmittelbar korrigieren und für eine Anpassung sammeln und abspeichern lassen. Da es sich bei dieser Anwendung um die Verarbeitung firmensensibler Daten handelt, soll auf die Verwendung von Modellen mit externer Einspeisung oder Servern verzichtet werden. Zudem steht nur eine begrenzten Menge an Trainingsdaten zur Verfügung. Diese Rahmenbedingungen schließen den Einsatz großer, rechenintensiver Transformer-Sprachmodelle im Rahmen dieser Arbeit aus und erfordern stattdessen kompakte ML-Ansätze. Die technischen Einschränkungen, denen ein ML-Modell in dieser Arbeit unterliegt, bestehen aus einer Hardware mit 16 GB RAM, einem Intel Core i5-13400 Prozessor und einer Intel UHD Graphics 730 Grafikkarte, welche gleichzeitig die Ziel- und auch Testumgebung darstellen. Resultierend lässt sich die Problembeschreibung dieser Masterarbeit folgendermaßen formulieren:

„Es ist kein Verfahren bekannt, welches unter gegebenen Einschränkungen hinsichtlich Rechenleistung und Datenverfügbarkeit in der Lage ist, natürlichsprachliche Testanweisungen zur Validierung sicherheitskritischer Systeme zuverlässig automatisiert den korrekten Bedienelementen für eine strukturierte Testausführung zuzuordnen. “

Auf Basis der Problembeschreibung werden im folgenden Kapitel zugehörige Forschungsfragen abgeleitet.

1.3 Forschungsfragen

Eine flexible Zuordnung der Testspezifikationen zu den Bedienelementen lässt sich am effizientesten über die Semantik realisieren. Zu diesem Zweck muss die Testanweisung in ein numerisches Repräsentationsformat, ein Embedding [35], umgewandelt werden. Eine anschließende Mehrklassen-Klassifikation ermöglicht eine strukturierte Zuordnung der numerisch dargestellten Testanweisungen zu den Bedienelementen. Die Klassen entsprechen dabei den E3-Namen der Bedienelemente. Ohne diesen Schritt wäre eine automatisierte, konsistente Interpretation und Zuordnung des Textes nicht möglich. Die E3-Namen werden weiterführend von dem bestehenden prototypischen System zur Erstellung einer Testsequenz verwendet. Da es sich um FuSi-Tests handelt, muss zudem sichergestellt werden, dass die Ergebnisse der Zuordnung der Bedienelemente vertrauenswürdig sind. Entsprechend sind Maßnahmen nötig, um die Validität der Ergebnisse nach der Sicherheits-DIN-Norm EN 50657 zu verifizieren. Die jeweiligen Bestandteile dieses Verfahrens, die Methode der Text-Repräsentation und der Klassifikator, müssen im Zuge der Arbeit ermittelt werden und unterliegen den genannten Einschränkungen. Eine entsprechende Forschungsfrage (FF) lässt sich wie folgt formulieren:

FF1: „Mit welcher ML-Methode lassen sich Anweisungen für FuSi-Tests effizient definierten Bedienelementen zuweisen, wenn der Rahmen des Verfahrens in Bezug auf Ressourcen und Datenmengen eingeschränkt ist? “

Da es bereits ein System gibt, welches den Prozess der teilautomatisierten Testdurchführung umsetzt, muss eine Schnittstelle erstellt werden, über welche das zu erstellende Zuordnungs-Verfahren eingebunden wird. Zum Zweck der Wartung und Nutzbarkeit soll das unter Berücksichtigung der benötigten Ressourcen möglichst effizient erfolgen. Zudem soll die Möglichkeit bestehen, in Zukunft weitere Anpassungen der ML-Verfahren zu integrieren. Dies schließt auch die Einbindung leistungsfähigerer Modelle mit ein. Somit lässt sich eine weitere Forschungsfrage ableiten:

FF2: „Wie lässt sich das erstellte System effizient und unter Berücksichtigung der Aspekte zur Softwarequalität und zukünftigen Optimierungen in den bestehenden Prototypen integrieren? “

1.4 Forschungsmethodik und Aufbau der Arbeit

Die Forschungsmethodik richtet sich strukturell nach der Nunamaker-Methode [30]. Nach dieser Methode basiert der grundlegende Aufbau einer wissenschaftlichen Arbeit auf einer klaren Definition der Forschungsproblematik, aus welcher sich Forschungsfragen und Forschungsziele ableiten. Dieser Aufbau bestimmt die Struktur der Arbeit, aus welcher sich die Kapitel *Stand der Technik und theoretische Grundlagen*, *Modellierung*, *Integration* und *Evaluation* ergeben.

Das erste Kapitel nach der Einleitung behandelt den technischen Stand und die Grundlagen der semantische ML-Methoden, die Aufgrund der physikalischen Einschränkungen für die Lösung der ersten Forschungsfrage geeignet sind. ML-Modelle, die hierfür in Frage kommen, umfassen zum Einen Text-Transformer zur Umsetzung der semantischen Information und zum Anderen Klassifikatoren für die Zuweisung der Texte zu den Bedienelementen. Die Grundlagen dazu werden in Unterabschnitt 2.1 und Unterabschnitt 2.2 erläutert. Des Weiteren wird das bereits bestehende prototypische System vorgestellt. Es beschreibt die Umgebung und den grundlegenden Kontext, in welches das intelligente System eingebettet wird. Dies schließt eine Vorstellung der Testspezifikationen, der Struktur der Testanweisungen und die Funktionsweise der Testsequenzen mit ein.

Das zweite Kapitel beschreibt die vollumfängliche Architektur und die Modellierung des zu erstellenden Systems. Dies beinhaltet eine konkrete Planung des Zuordnungsmodell in Unterabschnitt 3.1 sowie eine konkrete Trainings-Strategie Unterabschnitt 3.1.5. Diese ist maßgeblich für die Flexibilität und den Erfolg des Klassifikations-Modells und stellt eine zentrale Herausforderung dar. Zudem wird in Unterabschnitt 3.2 die Einbettung in den bestehenden Prototypen geplant und strukturiert. Der Fokus liegt auf einer guten Integrierbarkeit und Modularität, sodass sich das Modell-System auch in andere ähnliche Systeme integrieren lässt.

Im Abschnitt 4 erfolgt die praktische Umsetzung der Modellierung als ein prototypisches System. Die zu erstellende Modell-Kombination wird in dieser Phase eingerichtet sowie alle Voraussetzungen für ein Text-Embedding und eine anschließende Klassifikation erzeugt. Es folgt das Training des Systems mit den vorbereiteten Trainingsdaten. Im Anschluss an ein erfolgreiches Training wird die Anwendbarkeit außerhalb des Ziel-Systems in Zusammenhang mit Testdaten geprüft, bevor das erstellte System integriert wird. Die Integration erfolgt nur bei erfolgreicher Umsetzung des zuvor erstellten Modells.

Die Evaluierung in Abschnitt 5 testet die Effizienz des erstellten ML-Modells. Mit geeigneten und möglichst variierenden Testfällen wird die Eignung des erstellten Systems auf die Problemstellung der Arbeit herausgestellt und bewertet. Die Bestandteile der Evaluierung bestehen dabei nicht nur aus den analysierenden Tests sondern auch in der Vorbereitung der Testdaten. Deren Qualität ist maßgeblich

für die Aussagekraft der Evaluierungsergebnisse. Die Modell-Tests selbst werden unter verschiedenen Konditionen ausgeführt, um die Performance-Bandbreite des Modells zu erfassen:
- Variierende Längen an Testschritten
- Verschiedene Synonyme und Formulierungen der selben semantischen Bedeutung
- Integration von Schreibfehlern
Dies soll mögliche Schwankungen in der Effizienz identifizieren und hervorheben, unter welchen Bedingungen das Modell optimal arbeitet oder Schwachstellen aufweist. Die Testdaten werden auf unterschiedliche Eigenschaften hin untersucht:
- Zusammensetzung des Merkmalsvektors
- Ermittlung der optimalen Kernel-Funktion
- Performance (F1, Precision, Recall)
- Ressourcenbedarf (RAM-Speicher, CPU-Auslastung)
- Modell-Qualität (Genauigkeit, Trainingsverlauf)
- Vergleichstests
Auf Basis der Evaluierungs-Ergebnissen werden entsprechende Verbesserungsmöglichkeiten herausgestellt und diskutiert. Im Anschluss erfolgt in Abschnitt 6 ein Ausblick auf weiterführende Herausforderungen und Nutzungsmöglichkeiten des entwickelten Modells.

2 Stand der Technik und theoretische Grundlagen

Das nachfolgende Kapitel umfasst einen detaillierten Überblick über die theoretischen Grundlagen dieser Arbeit und wie die jeweiligen Methodiken in der aktuellen Forschung positioniert sind. Zunächst werden die Grundprinzipien im Kontext der ML-gestützten semantischen Interpretation dargelegt. Diese umfassen die Basis der natürlichen Sprachverarbeitung (engl. natural language processing) (NLP), der Transformer-Modelle sowie deren Kontext und weiteren Verarbeitungsmöglichkeiten. Zudem wird speziell auf das Modell *all-MiniLM-L6-v2* eingegangen, da dieses das Hauptuntersuchungsobjekt der Arbeit darstellt. Weiterführend erfolgt eine Einführung in ML-gestützte Klassifikation, welche ebenfalls in dieser Arbeit Anwendung finden wird. Zuletzt erfolgt die Darstellung der technischen Hintergründe, die den Kontext für die Problemstellung und das technische Umfeld für die Anwendung bilden (FF2). Sie zeigen zudem die Limitierung der Ressourcen auf, welche die Nutzung eines kleinen Modells erfordern und die Restriktionen für ein ML-Setup darstellen.

2.1 Vorstellung von Methoden zur semantischen Interpretation

Im Folgenden werden die Grundlagen des maschinellen Lernens, der aktuelle Forschungsstand und deren Anwendung in der semantischen Interpretation von Texten vorgestellt. Dies beinhaltet eine grundlegende Betrachtung der Verarbeitung von natürlicher Sprache und zugehörige Methoden des maschinellen Lernens. Zudem wird auf sogenannte Transformer Modelle [39] eingegangen, die in der Verarbeitung im NLP eine elementare Rolle spielen. Speziell das Modell *all-MiniLM-L6-v2* erfährt am Ende des Kapitels besondere Aufmerksamkeit.

2.1.1 Maschinelles Lernen

Definition 2.1. *ML [6] bezeichnet ein Verfahren, bei dem Modelle aus vorhandenen Trainingsdaten Muster, Strukturen und funktionale Zusammenhänge ableiten, ohne dass diese explizit programmiert werden. Ziel ist es, anhand gelernter Merkmalsrepräsentationen neue, unbekannte Eingaben möglichst korrekt vorherzusagen oder zu klassifizieren.*

Als Input für ML-Modelle dienen Merkmalsvektoren, die messbare Eigenschaften der gewünschten Größe in numerischer Form repräsentieren. Dieses Merkmal der Lernfähigkeit spiegelt zudem den Unterschied zur klassischen Programmierung dar. Dort werden Regeln explizit in Algorithmen vorgegeben. Anwendungsmöglichkeiten für ML-Modelle bestehen unter anderen in Vorhersagen von Zahlenwerten, Klassifikationen, Entscheidungsbäumen oder in neuronalen Netzen zur komplexen Musterbildung.

Wie in Abbildung 1 dargestellt wird, ist ML eine Unterkategorie der Künstlichen Intelligenz (KI) [37]. KI verkörpert das Konzept und ML repräsentiert eine

Methode für das von KI genutzte Lernen aus Daten. Es gibt dabei drei verschiedene Arten von Lernverfahren [7], die sich jeweils in ihrem Umgang mit den Trainingsdaten unterscheiden: überwachtes Lernen, unüberwachtes Lernen und bestärktes Lernen.

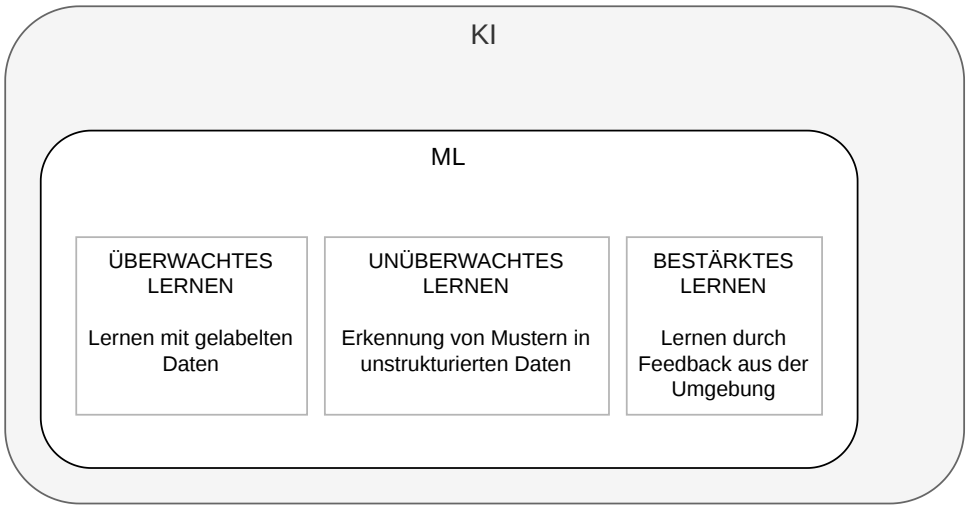

Abbildung 1: Übersicht über die Einordnung und Lernmethoden des ML

Überwachtes Lernen Überwachtes Lernen [7] (engl. supervised learning) basiert auf einem vordefinierten Output und lernt Zusammenhänge anhand gelabelter Daten. Im Kontext einer Klassifizierung sind eigene Klassen definierbar und bilden den Rahmen der Klassifikation. Der Input wird diesen Klassen auf Basis der gelernten Zusammenhänge zugewiesen. Neue Klassen lassen sich somit nur durch erneutes Anlernen der Daten integrieren.

Unüberwachtes Lernen Der Gegensatz zum überwachten Lernen besteht im unüberwachten Lernen [7] (engl. unsupervised learning). Es ist kein Output vordefiniert, sodass Klassen oder Datengruppen eigenständig vom Modell durch Clustering erstellt werden. Dies bietet eine einfache Integrierbarkeit neuer Datengruppen.

Bestärktes Lernen Bestärktes Lernen (engl. reinforcement learning) wird von Sutton und Barto [38] unter anderen als trial-and-error Methode beschrieben. Dies bezieht sich auf den Umstand, dass das Modell durch Ausprobieren lernt, was zur Belohnung oder Bestrafung für entsprechendes Verhalten führt. Die Entscheidungen des Modells werden zufällig getroffen und auf Basis des Feedbacks angepasst. Die Umgebung, in welcher das Modell trainiert wird, spielt eine zentrale Rolle, da diese ausschlaggebend für das Feedback und somit den Lernerfolg ist.

Modellgröße Die Arbeit mit kleinen ML-Modellen, wie sie in dieser Arbeit angestrebt wird, erfordert eine ausgewogene Balance zwischen Ressourcenverbrauch und Modellleistung wie in Abbildung 2 dargestellt. Heydari und Mahmoud [17] stellen diesbezüglich sowohl allgemeine als auch bereichsspezifische Hürden im Bereich der sogenannten TinyMLs vor. Die allgemeinen Herausforderungen umfassen Hardwareanforderungen wie begrenzte Speicher- und Rechenkapazität. Dies begrenzt folglich jedoch auch die Komplexität der Modelle, was die Balance zwischen Modellgröße und Vorhersagequalität (siehe Abbildung 2) zu einem zentralen Punkt werden lässt. Auch der Energieverbrauch ist eine zu beachtende Größe, da die Nutzung auf batteriebetriebenen Geräten einen Einfluss auf den Stromverbrauch haben. Wie sich die genannten Punkte in dem erstellten Modell dieser Arbeit verhalten, wird in der Evaluierung in Unterabschnitt 5.5 untersucht.

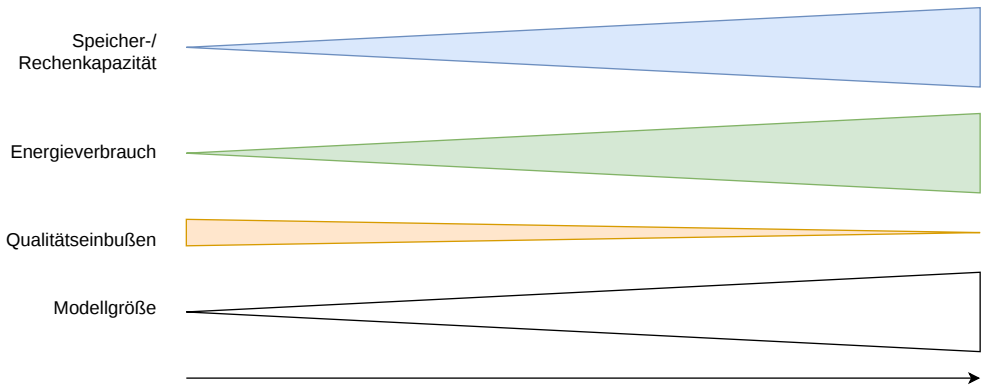

Abbildung 2: Korrelation der ML-Modellgröße mit der Ressourcenauslastung und Qualitätseinbußen

2.1.2 Mechanismus des überwachten Lernens

Der Mechanismus, der dafür sorgt, dass das Modell aus gelernten Daten fundierte Vorhersagen trifft, besitzt seine Grundlage im Lernprozess und wird im Folgenden anhand linearer Regression [44] beschrieben. Beim überwachten Lernen liegt ein Datensatz

$$D = \{(x^i, y^i)\}_{i=1}^{m} \tag{1}$$

vor, wobei $x \in \mathbb{R}^n$ der numerische Merkmalsvektor der Eingabe und $y \in \mathbb{R}$ der zugehörige Zielwert ist. Das Ziel besteht darin, eine Funktion $f : \mathbb{R}^n \to Y$ zu bestimmen, sodass

$$f(x) = y. \tag{2}$$

Der Kern des Modells besteht dabei aus einer parametrisierten Funktion der Form

$$f(x) = wx + b. \tag{3}$$

Die variablen numerischen Größen werden in Gewichte $w \in \mathbb{R}^n$ und Bias $b \in \mathbb{R}$ unterschieden und variieren in ihrer Anzahl je nach Größe des Modells. Bei einem Klassifikationsproblem [44] handelt es sich bei y um diskrete Werte, sodass hier eine Wahrscheinlichkeitsverteilung über alle Klassen (Ausgaberaum Y) erfolgt. Im Zuge des Trainings wird eine möglichst exakte Anpassung der Parameter angestrebt, um die Genauigkeit der Vorhersagen zu erhöhen. Die Eingabe des Modells besteht im Allgemeinen aus mehreren Merkmalen

$$x = (x_1, x_2, ..., x_n). \tag{4}$$

Im linearen Fall ergibt sich eine Modellfunktion als Linearkombination dieser Merkmale und deren Gewichtungen:

$$f(x) = w_1x_1 + w_2x_2 + ... + w_nx_n + b = w^Tx + b \tag{5}$$

Der Vektor x repräsentiert dabei einen Punkt im n-dimensionalen Vektorraum, dessen Dimension der Anzahl der Merkmale entspricht.

Das Ziel des Trainings besteht darin, die Parameter w und b so zu bestimmen, dass die daraus entstehende Trennfläche die Datenpunkte der unterschiedlichen Ziel-Klassen möglichst gut voneinander abgrenzt. Um eine optimale Parameterwahl zu ermöglichen, werden mehrere Lernzyklen durchlaufen. Das Modell startet dabei mit zufälligen Parametern und berechnet die Differenz $L \in \mathbb{R}$ (engl. loss) der Vorhersage $p \in \mathbb{R}$ zum Zielwert y mittels einer Verlustfunktion [44]:

$$L = (p - y)^2 \tag{6}$$

Dies ist ein Maß dafür, wie sehr die Vorhersage mit den gewählten Parametern von dem gegebenen Ziel-Label abweicht. Aus der Verlustfunktion werden die Gradienten berechnet, um zu bestimmen, wie w und b optimiert werden müssen. Die Parameter werden entsprechend korrigiert. Ein Lernzyklus wird dabei so lange wiederholt, bis die Differenz L nicht mehr sinkt. Eine grafische Übersicht ist mit Abbildung 3 gegeben.
Das folgende Beispiel soll den Lernprozess verdeutlichen. Die Fahrzeuge Fahrrad, Motorrad und Auto sollen anhand ihrer Räderzahl und Motorisierung klassifiziert werden. Somit bestehen die Zielklassen mit der folgenden Kodierung Auto = (1,0,0), Motorrad = (0,1,0), Fahrrad = (0,0,1):

$$Y = \{Auto, Motorrad, Fahrrad\}. \tag{7}$$

Die Merkmale beinhalten die Reifenzahl

$$r \in \{2, 4\} \tag{8}$$

und der Motorisierung

$$m \in \{0, 1\}. \tag{9}$$

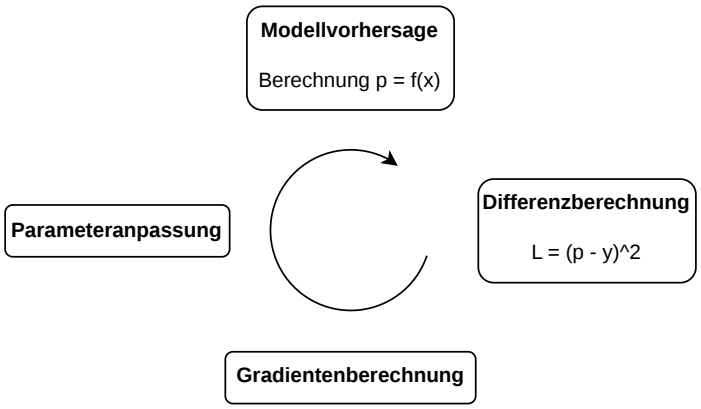

Abbildung 3: Lernzyklus des überwachten Lernens eines ML-Modells

Der Merkmalsvektor hat folglich die Dimension $d = 2$ mit

$$x = (r, m)^T \in \mathbb{R}^2 \tag{10}$$

Mit gelabelten Beispieldaten bestimmt das Modell die Parameter w und b. Ein Beispielinput $x_b = (2, 1)$ ergibt eine Vorhersage von

$$f(x_b) = (0.1, 0.8, 0.1), \tag{11}$$

was der Klassifikation des Fahrzeugs als ein Motorrad $y = (0, 1, 0)$ entspricht. Die Differenz

$$L = (f(x_b) - y)^2 = (0.8 - 1)^2 = 0.04 \tag{12}$$

zwischen Vorhersage und korrektem Zielwert ist gut aber nicht perfekt. Der Wert lässt sich über eine Anpassung der Parameter verbessern und erhöht die Wahrscheinlichkeit für eine korrekte Klasse mit weiteren Lernzyklen.

Der Lernprozess lässt sich überprüfen, indem die Modellgenauigkeit anhand der Trainings- und Testdaten über den Trainingsverlauf aufgezeichnet wird. Anhand der Kurvenform lässt sich erkennen, ob das Modell überangepasst ist (engl. overfitting) oder gut generalisiert. Dies wird in der Evaluierung in Unterabschnitt 5.6.2 weiter vorgestellt und analysiert.

2.1.3 Natürliche Sprachverarbeitung

Die natürliche Sprachverarbeitung (engl. Natural Language Processing (NLP) [11] ist ein Forschungs- und Anwendungsfeld innerhalb der KI und umfasst alle Methoden für die Analyse und Interpretation menschlicher Sprache. Das Ziel besteht darin, ein ML-Modell bzw. System so zu gestalten, dass es Texte und gesprochene Sprache semantisch versteht und verarbeitet. Somit soll auf eine sprachliche Eingabe auf eine sinnvolle Art und Weise reagiert werden können. Typische Anwendungen und Methoden innerhalb des NLP bestehen in der Erzeugung von Embeddings (siehe Unterabschnitt 2.1.5), der semantischen Ähnlichkeitssuche (siehe Unterabschnitt 2.1.4, der automatisierten Textzusammenfassung, Frage-Antwort-Systemen oder der Klassifikation von Texten beispielsweise mit Support Vector Machines (SVM) (siehe Unterabschnitt 2.2.2).

2.1.4 Semantische Textähnlichkeit

Definition 2.2. *Die semantische Ähnlichkeit von Texten (engl. Semantic Text Similarity (STS) [24]) bewertet den semantischen Grad der Übereinstimmung zwischen zwei Texten unabhängig von ihrer sprachlichen Formulierung.*

Das bedeutet, dass für die gegebenen zwei Sätze

$$s_1, s_2 \in \mathcal{S} \tag{13}$$

eine Funktion

$$f(s_1, s_2) \in [0, 1] \tag{14}$$

ermittelt werden soll, die ausdrückt, wie ähnlich sich die beiden Texte inhaltlich sind. Das Resultat ist der Grad der Übereinstimmung auf einer Skala $[0, 1]$ und keine binäre Aussage $\{0, 1\}$, ob die Texte verschieden sind oder nicht. Zur Berechnung der semantischen Ähnlichkeit kommen verschiedene Methoden zum Einsatz. Neben Verfahren des maschinellen Lernens und semantischen Modellen werden auch statistische Verfahren und Vektorraumdarstellungen (siehe Unterabschnitt 2.1.5) verwendet und ergänzen somit grundlegende Funktionen von NLP (Unterabschnitt 2.1.3). Dies ermöglicht einen Vergleich von sowohl kurzen als auch umfangreichen Texten hinsichtlich ihrer Bedeutung.

Es gibt verschiedene Methoden, um STS im NLP Kontext umzusetzen. Drei Hauptkategorien werden von Chandrasekaran und Vijay [10] vergleichend gegenübergestellt und sind in Abbildung 4 zusammengefasst: wissensbasierte STS, korpusbasierte STS und auf neuronalen Netzen aufbauende STS.

Wissensbasierte STS Die wissensbasierte semantische Ähnlichkeit greift auf Wissensquellen wie Ontologien, Lexika oder WordNet [26, 13] zurück. Der Vorteil liegt in der unmittelbaren Nutzung von definierten semantischen Bedeutungen, jedoch ist die Qualität stark von der verwendeten Wissensbasis abhängig. Die

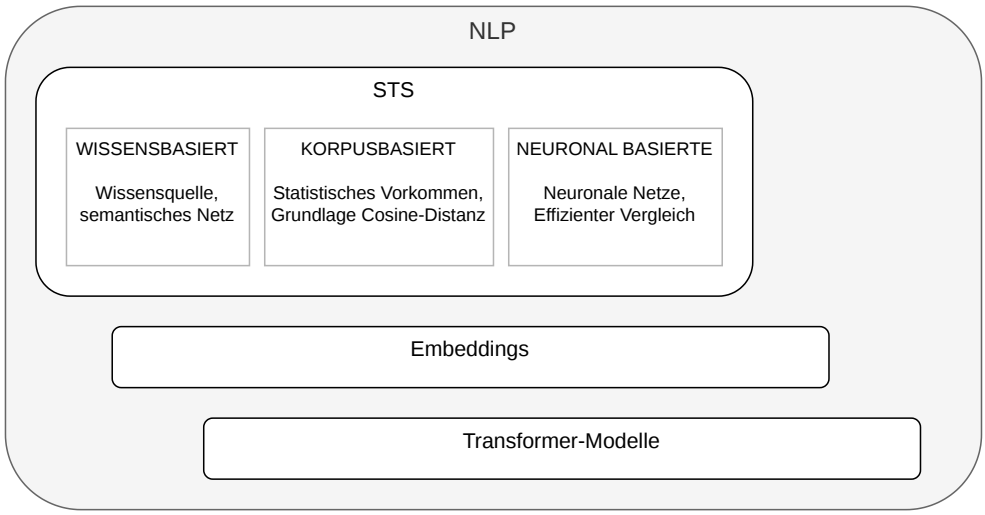

Abbildung 4: Übersicht über die Einordnung und Kategorien der STS

Anwendung auf unbekannte Wissensdomänen ist somit ohne domänenspezifische Wissensdaten nur eingeschränkt nutzbar. Beispielsweise ist die semantische Ähnlichkeit in WordNet zwischen dem Wort „Hund“ und „Tier“ sehr hoch, da „Hund“ ein Unterbegriff von „Tier“ ist. Die Wörter „Hund“ und „Auto“ weisen im Gegensatz dazu keine strukturelle Beziehung und somit auch keine semantische Ähnlichkeit auf.

Korpusbasierte STS Im Gegensatz dazu arbeitet der korpusbasierte Ansatz mit dem statistischen Auftreten von Wörtern in ähnlichen Kontexten. Sei dabei die Wahrscheinlichkeit des Vorkommen des Wortes w gegeben mit $P(w)$ und dem gemeinsamen Auftreten in einem Kontext c mit $P(w, c)$, dann ergibt sich die punktweise Transinformation [20] (engl. pointwise mutual information (PMI))

$$PMI(w, c) = log \frac{P(w, c)}{P(w)P(c)}. \tag{15}$$

Die Ähnlichkeit zwischen den Texten t_1, t_2 lässt sich als Aggregation darstellen

$$STS_{corp}(t_1, t_2) = \frac{1}{|t_1| \cdot |t_2|} \sum_{w \in t_1} \sum_{w \in t_2} PMI(w, c) \tag{16}$$

Die eigentliche Bedeutung der Wörter wird bei dieser Methode nicht beachtet. Sie bietet jedoch eine größere Flexibilität hinsichtlich der Anwendungsbreite, solange eine ausreichend qualitative Datengrundlage vorhanden ist. Dieser Ansatz dient als Grundlage für viele semantische Distanzmaße zwischen Wort-Embeddings [27]

(Definition in Unterabschnitt 2.1.5). Unter anderen bildet er die Basis für ein heute weit verbreitetes Maß, der Kosinus-Ähnlichkeit (engl. cosine distance)

$$\mathrm{sim}_{cos}(x, y) = \frac{x \cdot y}{|x| \cdot |y|} \tag{17}$$

mit $x, y \in \mathbb{R}^n$ als Vektorrepräsentationen der Texte. Ein Beispiel für diesen Ansatz besteht in der Ähnlichkeit der Wörter „Arzt " und „Doktor". Beide Wörter werden häufig in ähnlichen Kontexten (z.B. Krankenhaus) verwendet. Statistische Modelle ordnen sie daher als semantisch ähnlich ein.

Neuronal basierte STS Das Aufkommen neuronaler Netze [2] unterstützt die Erfassung, Verarbeitung und Modellierung komplexer semantischer Abhängigkeiten zwischen Vektorrepräsentationen. Eine neuronale STS lässt sich über die Kosinus-Ähnlichkeit mit einem Encoder (parametrisiert durch Gewichte θ)

$$E_\theta : T \to \mathbb{R}^d \tag{18}$$

für die Texte t_1, t_2

$$x = E_\theta(t_1), y = E_\theta(t_2) \tag{19}$$

folgendermaßen formulieren:

$$STS_{nn}(t_1, t_2) = \frac{x \cdot y}{|x| \cdot |y|} \tag{20}$$

Alternativ als ein trainiertes Regressionsmodell

$$STS_{nn}(t_1, t_2) = h_\phi([x; y; |x - y|; xy]) \tag{21}$$

mit dem neuronalen Vorhersagemodell h_ϕ und einer elementweisen Multiplikation .

Mit den Transformer-Modellen [39] (vorgestellt in Unterabschnitt 2.1.6) kommt zusätzlich ein Modellansatz hinzu, der kontextabhängige Satzrepräsentationen ermöglicht. Damit können Abhängigkeiten und Synonyme präzise erfasst werden, was nach aktuellem Stand der Forschung die besten Ergebnisse liefert. Dabei sind jedoch erhebliche Abstriche hinsichtlich Ressourcenbedarf und Modelltransparenz hinzunehmen. Als Beispiel für diese Methode besitzen die Sätze „Das Essen wurde vom Hund gefressen" und „Das Haustier nimmt Nahrung zu sich" unterschiedliche Wörter und Satzstrukturen, jedoch bedeuten beide „Der Hund frisst" und sind somit semantisch ähnlich. Im Kontext dieser Arbeit wird STS zur qualitativen Aufwertung des Merkmalsvektors für die Klassifikation verwendet, dessen Aufbau in Unterabschnitt 3.1.3 erläutert wird.

2.1.5 Embeddings

Definition 2.3. *Ein Embedding [35] ist eine Abbildung*

$$f : \mathcal{X} \to \mathbb{R}^n, \tag{22}$$

welche nichtnumerischen Objekte $x \in \mathcal{X}$ in einen n-dimensionalen, reellen Vektorraum überführt.

Ziel eines Embeddings ist es, semantische Eigenschaften der Objekte so zu repräsentieren, dass deren Beziehungen zueinander im Vektorraum widergespiegelt werden (siehe Abbildung 5). Die Abstände zwischen den Punkten lassen dabei Aussagen über die semantische Ähnlichkeit zu. Sätze und Wörter mit verwandter Bedeutung befinden sich im Raum näher beieinander als solche, die eine größere inhaltliche Distanz haben. Embeddings stellen somit die Verbindung zwischen dem Verständnis der menschlichen Sprache und der mathematischen Verarbeitbarkeit her. Erzeugt werden sie mithilfe von Transformer-Modellen (vorgestellt in Unterabschnitt 2.1.6, die speziell für die Verarbeitung von Texten eingesetzt werden.

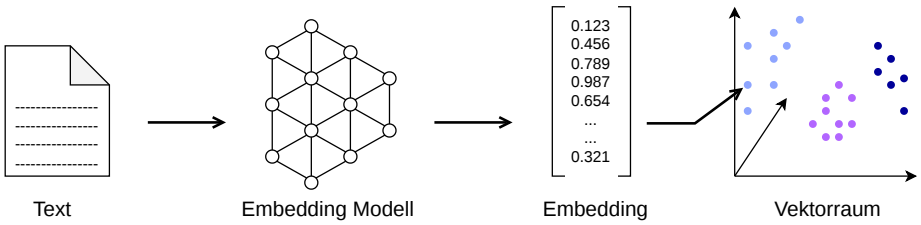

Abbildung 5: Erzeugung und Repräsentation von Embeddings

Die Einsatzmöglichkeiten spielen eine große Rolle im Kontext der natürlichen Sprachverarbeitung (Unterabschnitt 2.1.3). Vor allem, wenn es um Anwendungen zur Textklassifikationen, Clustern von Informationen oder Übersetzungen geht und in der zuvor genannten STS (Unterabschnitt 2.1.4). Generell wird zwischen drei Arten von Embeddings unterschieden, welche in Abbildung 6 gegenüber gestellt sind: häufigkeitsbasierte Embeddings, vorhersagebasierte Embeddings, kontextabhängige Embeddings.

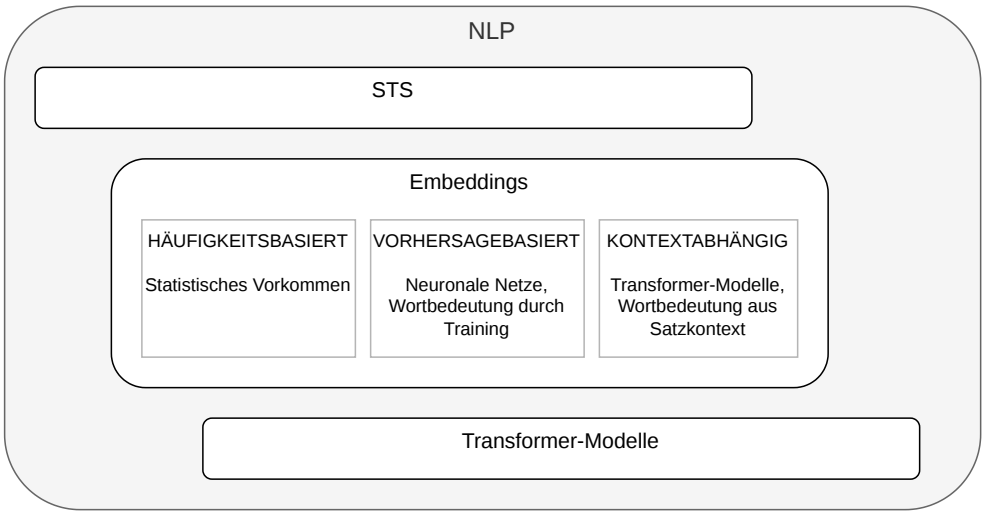

Abbildung 6: Übersicht über die Einordnung und Kategorien von Embeddings

Häufigkeitsbasierte Embeddings Die häufigkeitsbasierten oder auch traditionellen Embeddings stellen sprachliche Zusammenhänge über statistische Verfahren dar. Dabei wird ermittelt, wie oft ein Wort in einem Text vorkommt und mit welchen anderen Wörtern es zusammen erscheint. Das Vorkommen V lässt sich beispielsweise über die Anzahl der Dokumente N und der Frequenz des Wortauftretens $df(w)$ bestimmen mit

$$V = log \frac{N}{df(w)} \tag{23}$$

Diese Methode erzeugt dabei jedoch große und schwer zu handhabende Vektoren.

Vorhersagebasierte Embeddings Vorhersagenbasierte Embeddings basieren auf neuronalen Netzen, welche eine Wort-Repräsentation mittels Training eines Modells erzeugt. Die Bedeutung eines Wortes ist nach Abschluss des Trainings festgelegt. Wörter werden mithilfe des erstellten Modells anhand ihres Kontextes vorhergesagt, dies besitzt jedoch den Nachteil, dass Bedeutungsunterschiede eines Wortes in Abhängigkeit des Kontextes nicht berücksichtigt werden. Beispielsweise lernt in Modell, dass „Auto“ häufig im Zusammenhang mit Wörtern wie „fahren“, „Straße“ oder „Motor“ auftritt. Dadurch erhält „Auto“ einen ähnlichen Vektor wie „Fahrzeug“, da das Modell das gemeinsame Auftreten mit einer hohen Wahrscheinlichkeit vorhersagt.

Kontextabhängige Embeddings Dieses Problem greifen kontextabhängige Embeddings auf. Die Wortbedeutung wird dynamisch aus dem Satzumfeld gebildet. Für die Erzeugung dieser Embeddings werden bevorzugt Transformer-Modelle ver-

wendet, welche vor allem im Bereich der NLP eingesetzt werden und vielfältige Aufgaben lösen können. Wie zuvor bei der neuronalen STS wird ein parametrisierter Encoder

$$E_\theta : T \to \mathbb{R}^d \tag{24}$$

verwendet. Ausgabe ist eine Matrix mit n Token-Vektoren. Das Satzembedding ergibt sich dann aus dem Mittelwert

$$Emb(t) = \frac{1}{n} \sum_{1=1}^{n} E_\theta(t)_i \tag{25}$$

Die beiden Sätze „Ich sitze auf einer Bank.“ und „Ich gehe zur Bank.“ haben beispielsweise komplett unterschiedliche Bedeutungen. Der erste Satz bezieht sich auf das Möbelstück und der zweite Satz auf das Finanzinstitut. Ein Transformer-Modell erzeugt hier abhängig vom Satzkontext zwei unterschiedliche Vektoren.

2.1.6 Transformer Modelle

Ein Transformer ist eine Architektur von Deep-Learning-Modellen, die 2017 eingeführt wurde [39] und die speziell auf einem Aufmerksamkeitsmechanismus (engl. attention) basiert anstatt auf rekurrenten Netzen (RNN). Sie wird speziell in der Textverarbeitung eingesetzt. Es bildet die Grundlage für viele im NLP verwendeten Modelle wie GPT und BERT. Die groben Bestandteile von Transformern [36] ist in Abbildung 7 zusammengefasst.

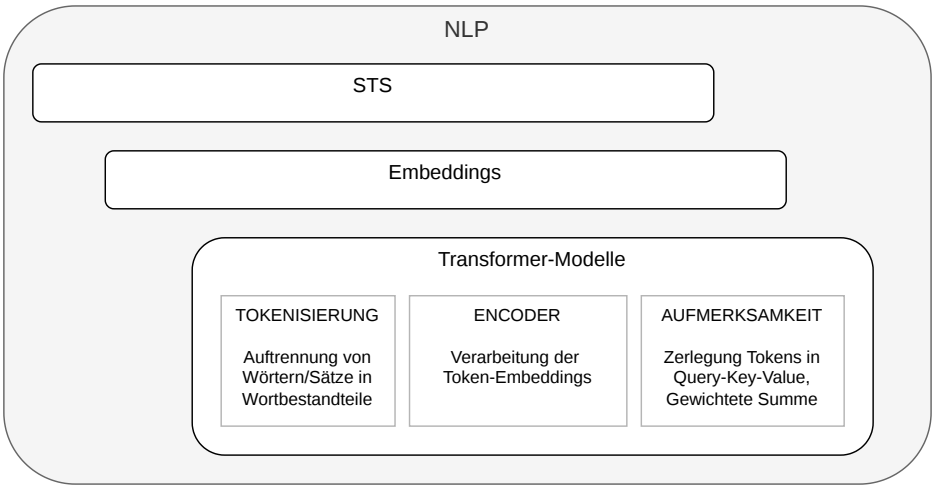

Abbildung 7: Übersicht über die funktionalen Bestandteile eines Transformer-Modells

Die Architektur eines Transformers besitzt dabei einen Aufbau, wie er in Abbildung 8 dargestellt ist. Die Verarbeitung der Eingabe erfolgt als Sequenz von Tokens.

Als *Tokens* [25] werden in NLP Wörter oder Wortbestandteile bezeichnet. Formal lässt sich ein Text t als Folge n diskreter Symbole modellieren:

$$t = (s_1, s_2, ..., s_n) \tag{26}$$

Eine Tokenisierungsfunktion

$$f_t : T \to V \tag{27}$$

weißt dabei jedem Text $t \in T$ eine Sequenz von Tokens $w_i \in V$ aus einem Vokabular V zugewiesen

$$f_t(t) = (w_1, w_2, ..., w_n) \tag{28}$$

Durch die Verwendung von Wortteilen lassen sich seltene oder komplexe Wörter abbilden und kontextuell einordnen. Die Verarbeitung wird durch eine Encoder vorgenommen. Dieser erhält für jedes Token ein Wort-Embedding

$$X_{wort} = \begin{pmatrix} E(w_1) \\ E(w_2) \\ ... \\ E(w_n) \end{pmatrix} \in \mathbb{R}^{n \times d} \tag{29}$$

für die Bedeutung sowie ein Positions-Embedding $X_{pos} \in \mathbb{R}^{n \times d}$ für die Reihenfolge.

Im Zuge des enthaltenen Aufmerksamkeitsmechanismus wird die Beziehung zwischen Tokens ermittelt. Dieser enthält eine Abfrage Q, einen Schlüssel K und eine Wert-Matrix V und lässt sich mit der Dimension d in der folgenden Form darstellen:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d}})V \tag{30}$$

Ein sogenannter *Multi-Head Attention* (siehe Abbildung 8) verarbeiten mehrere solche Funktionen parallel, die im Anschluss kombiniert werden. Dies fördert die Erfassung komplexer Zusammenhänge zwischen Tokens. Im Anschluss an den Aufmerksamkeitsmechanismus wird dessen Eingang auf die Ausgabe addiert und normiert, damit die Information nicht verloren geht. Im Anschluss verarbeitet ein *Feed-Forward Netzwerk* (FFN)

$$F : \mathbb{R}^{d_{in}} \to \mathbb{R}^{d_{out}} \tag{31}$$

die Information innerhalb des Tokens, welchem ebenfalls der Aufaddierungs-Mechanismus nachgestellt ist. Ein zusätzlicher Decoder erstellt bei generativen Modellen (*Encoder-Decoder-Modelle*) abschließend eine Ausgabesequenz bestehend aus Tokens. Bei Klassifikation- und STS-Transformatoren handelt es sich um *Encoder-only-Modelle*. Die Gesamt-Architektur besteht insgesamt aus mehreren identischen Schichten (engl. layers), wie es auch in Abbildung 8 abgebildet ist. Durch die Parallelisierung der Aufmerksamkeitsmechanismen benötigt das Training eine kürzere Zeit. Die Transformer bilden so die Grundlage für viele Modelle mit Bezug zur NLP.

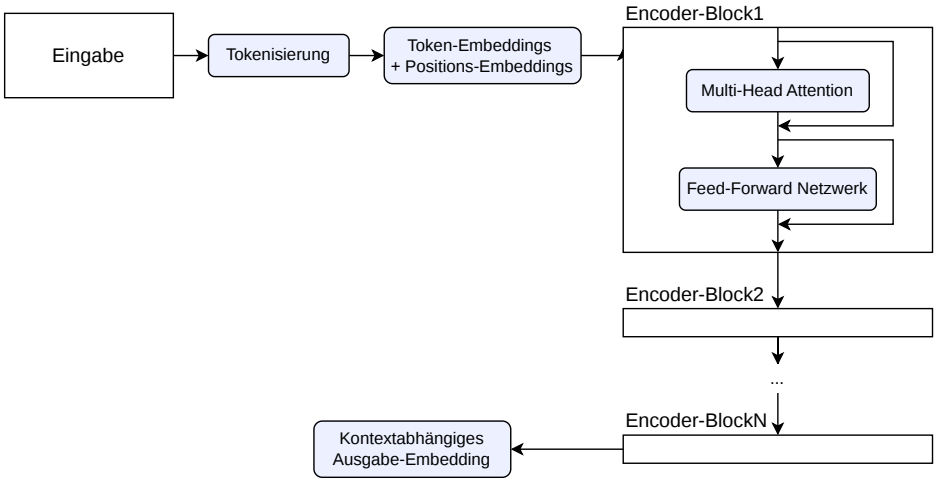

Abbildung 8: Architektur eines Transformer-Modells und dessen Verarbeitungsschritte

2.1.7 Sentence-Transformer all-MiniLM-L6-v2

Das *all-MiniLM-L6-v2* Modell [34] ist ein *Encoder-only* Sentence-Transformer, welches auf Kluster-Erzeugung (engl. Clustering), semantischem Ähnlichkeitsvergleich (engl. Semantic Similarity) und Suche spezialisiert ist. Aus eingegebenen Textabschnitten lassen sich Embeddings erzeugen, die unter Aufwendung geringer Ressourcen eine semantische Repräsentation der Eingabe darstellen.

In der Tabelle 1 werden die Eigenschaften des *all-MiniLM-L6-v2* aufgeführt. Im Vergleich zu anderen, ähnlichen Modellen wie all-distilroberta-v1 und all-mpnet-base-v2 [31] überzeugt *all-MiniLM-L6-v2* mit einer sehr guten Zeiteffizienz, wobei die semantische Repräsentation geringfügige Abstriche aufweist. Im Gegensatz dazu besitzen Modelle mit besserer Genauigkeit einen höheren Ressourcenverbrauch und größeren Speicherbedarf auf. Pavlyshenko und Stasiuk [31] kommen zu dem Schluss, dass *all-MiniLM-L6-v2* unter anderen zu den leistungsstarken Modellen der Bewertung gehört und somit geeignet für die Anwendung auf benutzerspezifische Daten ist. In Bezug auf diese Arbeit sorgen die vergleichsweise geringen Embedding-Dimensionen von 384 nicht nur für eine schnellere Embedding- und Vektordatenbank-Berechnung sondern auch weiterführend für eine schnellere Distanz-Berechnung während der STS. Aufgrund der geringeren Merkmalsvektor-Größe als Eingabe für das Klassifikationsmodell verringert sich hier neben dem Speicherbedarf ebenfalls sowohl die Trainingszeit als auch die die Zeit, in welcher eine Vorhersage getroffen wird (Inferenzzeit). Im Kontext einer nutzerinteraktiven Anwendung ist eine geringe Inferenzzeit dabei von entschei-

dender Bedeutung. Auch der Umstand, dass das *all-MiniLM-L6-v2* explizit auf CPU-Nutzung angepasst ist, passt sehr gut zu der Problemstellung dieser Arbeit.

Merkmal	Beschreibung
Architektur	MiniLM (kleines Transformer-Modell)
Anzahl Layer	6
Anzahl Parameter	22 Milliarden
Embedding Dimensionen	384
Max Tokens	256
Basismodell	MiniLM-L6-H384-uncased
Fine-tuning training pairs	1.2B

Tabelle 1: Modell-Eigenschaften und Daten des *all-MiniLM-L6-v2*

Im Vergleich zu anderen gleich großen Modellen kommt die Auswahl sehr auf den Anwendungszweck an. Für den Anwendungskontext einer ressourcenlimitierten Laptop-Umgebung ist eine stabile, vorhersagbar schnelle Inferenz entscheidend. Gegen andere lightweight-Modelle besteht das *all-MiniLM-L6-v2* aufgrund seiner Architektur und der effizienten Nutzung des Sweet-Sports von Embedding-Größe und Token-Limit. Die Mechanik der Modell-Erstellung, die Deep Self-Attention Distillation [16], bewirkt, dass die MiniLM nicht nur den Output des großen Basis-Modell lernt, sondern die komplette Struktur. Dies bringt einen erheblichen Vorteil gegenüber ähnlicher Modelle wie TinyBERT, DistilBERT oder ALBERT. So kann eine hohe Qualität der Vorhersagen trotz wenig Parameter erreicht werden. Die Kombination aus 22 Milliarden Parametern, einer Embedding-Dimension von 384 und 6 Layer erzeugt ein effizientes Verhältnis zwischen semantischer Tiefe und Geschwindigkeit. Modelle mit weniger Parametern, Embedding-Dimension oder Layer sind zwar schneller, aber besitzen eine geringere Qualität. Modelle mit einer höheren Embedding-Dimension und mehr Layer nehmen eine größere semantische Tiefe auf, sind dafür jedoch langsamer. Die hohe Nutzung des MiniLM Sentence-Transformers [34], mit konsistenten Implementierungen in PyTorch, TensorFlow und Weiteren spricht zudem für dessen Zuverlässigkeit und für eine hoher Nachfrage der Nutzer. Aus diesen Gründen, zusammen mit der Spezialisierung des Modells auf semantische Ähnlichkeit, wird das *all-MiniLM-L6-v2* Modells in dieser Arbeit verwendet.

Ursprünglich entstanden ist das *all-MiniLM-L6-v2* im Zuge einer Community Woche, die von HuggingFace organisiert wurde. Das Projekt bestand darin, das beste Embedding-Modell mit einer Milliarde Trainingspaaren zu trainieren, wofür auch geeignete Hardware zur Verfügung stand. Als Basis diente das Modell MiniLM-L6-H384-uncased [33], welches mit einem Umfang von circa 1.2 Billionen Datenpaaren optimiert (engl. fine-tuned) wurde.

2.2 Vorstellung von Zuordnungs-Methoden

Im Verlauf der Arbeit werden verschiedene Methoden untersucht, die es ermöglichen, Testschritte aus funktionalen Testanweisungen nicht-semantischen Bezeichnern zuzuordnen. Die Eignung der Methoden hängt von verschiedenen Faktoren, unter anderen der Komplexität der Testschritte und der Verfügbarkeit von Trainingsdaten ab. Da in diesem Fall eine limitierte Menge an Daten und Rechenressourcen zur Verfügung stehen, wird neben der Klassifikation mithilfe einer SVM [45] eine weitere Methode zur Zuordnung untersucht. Zudem besteht die Möglichkeit, eine Embedding-Datenbank zu nutzen, um die Wissensbasis des Sprachmodells zu erweitern. Vor allem bei internen Begriffen oder Bezeichnern, wie es in dieser Arbeit der Fall ist, stellt dies eine Vereinfachung der Kontextualisierung für einen semantischen Vergleich dar. Die Grundlagen der genannten Methoden werden im Folgenden vorgestellt.

2.2.1 ML-gestützte Klassifikation

In Unterabschnitt 2.1.1 wurden bereits das Grundkonzept des ML vorgestellt. Dies beinhaltet die Herstellung eines Zusammenhangs zwischen Input- und Outputdaten auf Basis von Trainingsdaten.

Definition 2.4. *Unter ML-Klassifikation [5] versteht man ein überwachtes Lernverfahren, bei dem aus einer Menge an Trainingsdaten eine Abbildung gelernt wird, die unbekannte Eingaben definierten Klassenlabels zuordnet.*

Die dabei verwendeten Daten zum Trainieren des Modells stammen aus einer Menge

$$(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}, \qquad i = 1, \ldots, n, \tag{32}$$

welche mit einer endlichen Menge diskreter Klassenlabels gepaart ist.

$$\mathcal{Y} = \{1, \ldots, K\}. \tag{33}$$

Ziel der Klassifikation ist es, eine Funktion

$$f : \mathcal{X} \to \mathcal{Y} \tag{34}$$

so zu bestimmen, dass die Klassen neuer Daten korrekt vorhersagt werden. So lernt das Modell anhand konkreter Beispiele, welche Merkmalsmuster zu welchen Klassen gehören und wie sich diese voneinander abgrenzen. Im Folgenden werden die zur Zeit gängigsten Methoden zur Klassifizierung mit ML vorgestellt.

Zunächst gibt es verschiedene Arten der Klassifikation [5]. Die binäre Klassifikation ist die einfachste Variante. Hier besteht die Zielmenge Y aus zwei Klassen:

$$y = \{0, 1\} \tag{35}$$

Mit dem Eingaberaum $X \subseteq \mathbb{R}^n$ ist eine binäre Klassifikation eine Abbildung

$$f : X \to Y. \tag{36}$$

Ein Beispiel ist eine Einteilung von E-Mails in Spam und Nicht-Spam. Die entsprechende Erweiterung besteht in der Mehrklassen-Klassifikation, wie sie auch im Kontext dieser Arbeit vorliegt. Hier gilt für die Zielmenge ein größerer Umfang:

$$Y = \{1, 2, ..., n\} \tag{37}$$

mit $n > 2$. Diese Zuordnung erfolgt dabei eindeutig und erfordert Modelle, die mehrdimensionale Entscheidungsgrenzen erzeugen. Methoden, um diese umzusetzen [5], sind vor allem Entscheidungsbäume, Ensemble-Verfahren, K-Nächster Nachbar (engl. K-Nearest Neighbor (K-NN)), Support Vector Machines und neuronale Netze.

Entscheidungsbäume Bei Entscheidungsbäumen (siehe Abbildung 9) werden die Daten rekursiv anhand von Merkmalen zerlegt. Die Baumstruktur repräsentiert die Klassifikationsregeln, wobei jeder Knoten eine Entscheidung darstellt und die Blätter die Klassen. Entscheidungsbäume sind leicht zu trainieren und gut interpretierbar, neigen jedoch zur Instabilität. Entscheidungsbäume eigenen sich daher in einer Ensemble-Nutzung, wobei mehrere Bäume trainiert und deren Entscheidungen über ein Mehrheits-Votum kombiniert werden. Zur Veranschaulichung eines Entscheidungsbaums lässt sich das Beispiel der Klassifikation von Fahrzeugen anhand der Radanzahl und Motorisierung aus Unterabschnitt 2.1.2 heranziehen. Die jeweiligen Entscheidungen, Kanten und resultierende Klassen sind in Klammern in Abbildung 9 hinzugefügt.

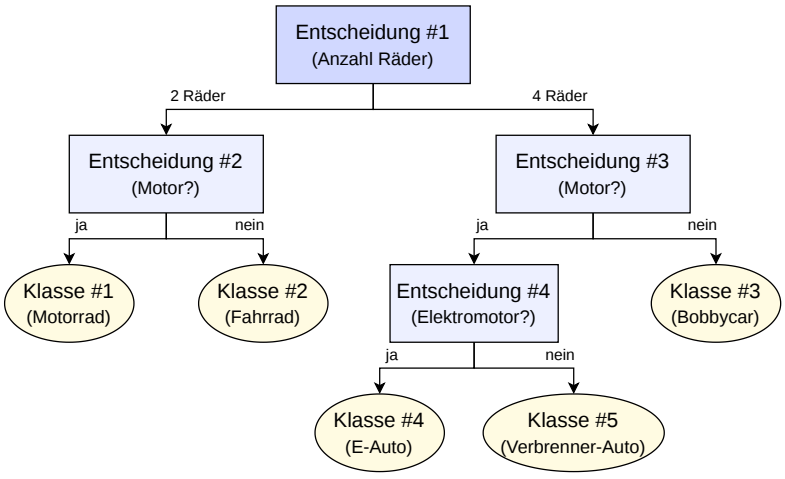

Abbildung 9: Übersicht über die Funktionsweise eines Entscheidungsbaums

K-NN Methode Neben Entscheidungsbäumen gibt es des Weiteren distanzbasierte Ansätze. Hierbei werden die Daten als Punkte in einem Vektorraum dargestellt und deren Distanzen beziehungsweise Positionierung zueinander bewertet. Bei K-NN erfolgt die Klassenzuordnung über die Zugehörigkeit der nächsten Nachbarn im Merkmalsraum (siehe Abbildung 10). Dafür ist keine Trainingsphase im klassischen Sinne erforderlich. Die gelabelten Daten werden im Merkmalsraum abgelegt und anhand derer werden neue Punkte den Nachbarn entsprechend zugeordnet. Dabei ist die Anzahl k der nächstliegenden Punkte relevant, die dafür einbezogen werden. In Abbildung 10 wird dieser Unterschied für k=3 und k=7 verdeutlicht. Bei k=3 würde der unbekannte Punkt der gelben Klasse zugeordnet werden, da mehr gelbe Punkte unter den drei nächsten Nachbarn sind. Bei k=7 verhält es sich jedoch umgekehrt. Hier sind mehr blaue Punkte enthalten, was eine Zugehörigkeit des unbekannten Punktes zur blauen Klasse bedeutet. Die Auswahl eines geeigneten k ist somit ein fundamentaler Bestandteil der Methode. Der Nachteil besteht darin, dass alle Trainingspunkte durchsucht und gespeichert werden müssen, was hohe Berechnungs- und Speicherkosten erzeugt.

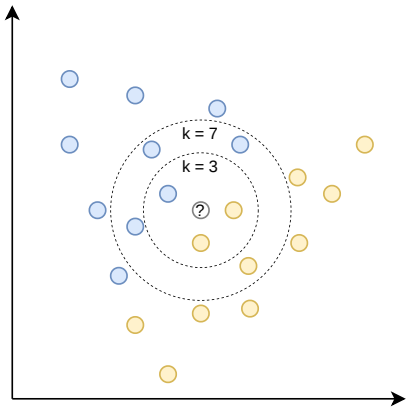

Abbildung 10: Übersicht über die Funktionsweise der K-NN-Methode

SVM Methode Eine Methode, die Training erfordert, ist die Support Vector Machine [45] (SVM). Diese ermittelt die optimale Trennlinie zwischen den Klassenwolken im Vektorraum (siehe Abbildung 11). Je nach Konfiguration ist es dabei auch möglich, nichtlineare Strukturen zu modellieren. Für eine effiziente Nutzung ist die Einstellung der Parameter entscheidend. Diese Methode der ML-gestützten Klassifikation wird näher in Unterabschnitt 2.2.2 betrachtet.

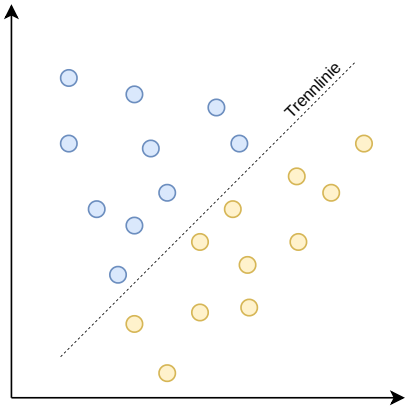

Abbildung 11: Übersicht über die Funktionsweise der SVM-Methode

Neuronales Netz Methode Eine weitere Möglichkeit im Bereich der trainierbaren Modelle besteht in der Nutzung von neuronalen Netzwerken [2]. Diese stellen die leistungsstärksten Verfahren dar und können auch komplexe Muster analysieren. Den Namen erhalten sie durch ihre Nachbildung des menschlichen Gehirns. Sogenannte Neuronen verarbeiten Eingaben unabhängig voneinander und lernen, Muster in Daten zu erkennen. Durch nichtlineare Funktionen sind auch komplexe Entscheidungsgrenzen einer Klassifikation lernbar. Jedoch erfordern sie einen hohen Trainingsaufwand und besitzen eine geringe Interpretierbarkeit.

2.2.2 Support Vector Machine

Definition 2.5. *Eine SVM [45] ist ein überwachtes Lernverfahren, welches eine Hyperebene zu Klassifikations- oder Regressionszwecken bestimmt.*

Das Ziel dabei ist es, eine Hyperebene zu ermitteln, die einen maximalen Abstand (Margin) zwischen den nächstgelegenen Datenpunkten verschiedener Klassen besitzt (siehe Abbildung 12). Dieser spezielle Datenpunkt wird Support-Vektor genannt und verleiht der SVM ihren Namen. Für $k \in \{1, ..., K\}$ Klassen werden dabei

k lineare SVMs gelernt

$$g_k(x) = w_k^T x + b_k, \tag{38}$$

mit welchen die Klassifikation

$$f(x) = argmax_k g_k(x) \tag{39}$$

erfolgt. Mathematisch handelt sich die Ermittlung der Hyperebene somit um ein Optimierungsproblem. Jede der k SVMs erzeugt dabei eine Hyperebene mit dem Ziel

$$w_k^T x + b_k = 0 \tag{40}$$

und einer möglichst großen Margin

$$m_k = \frac{1}{|w_k|}, \tag{41}$$

wobei w_k für die Optimierung möglichst klein gehalten wird.

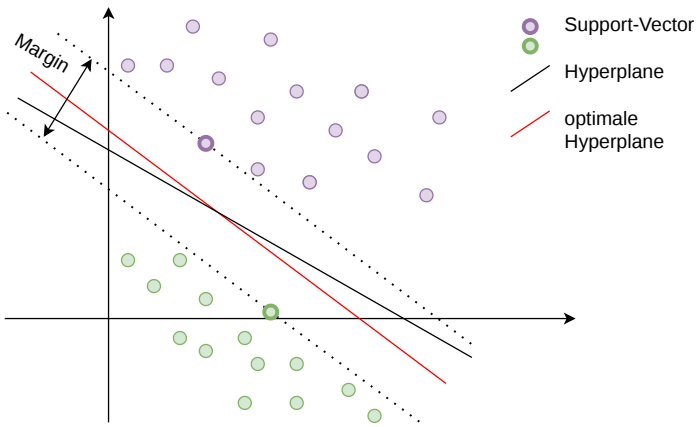

Abbildung 12: Zweidimensionale Darstellung einer SVM mit einer linearen Trennung der Klassenpunkte

Kernel Die Margin des Support-Vektors zur Hyperebene (siehe Abbildung 12) ist entscheidend für die spätere Generalisierbarkeit. Bei der Ermittlung der Hyperebene besteht allerdings das Problem der linearen Separierbarkeit. In einer Annahme sind Trainingsdaten linear separierbar, was in der Praxis jedoch oft nicht zutrifft. Die Lösung ist ein Mapping

$$\phi : \mathbb{R}^n \to \mathbb{R}^d, d \gg n \tag{42}$$

in einen höherdimensionalen Raum, in welchem die Daten separierbar werden. Anstatt diese Transformation explizit zu berechnen, wird eine sogenannte Kernel-Funktion eingesetzt. Diese bildet die Datenpunkte lediglich in einen höherdimensionalen Raum

$$k : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R} \tag{43}$$

ab, wo sie dann separierbar sind. Kernel bestimmen somit, wie gut Daten im Vektorraum getrennt werden und haben einen direkten Einfluss auf die Performance. Es gibt dabei keine klare Regel, welcher Kernel für welches Problem optimal ist. Um den besten Kernel für die eigene Anwendung zu finden, müssen experimentell verschiedene ausprobiert und evaluiert werden. Die Erkenntnisse von Zhang *et al.* [41] legen jedoch nahe, dass bei einer Textklassifikation ein linearer Kernel zu bevorzugen ist. Dies wird im Zuge der Evaluierung in Abschnitt 5 überprüft und ein passender Kernel ermittelt.

Hyperparameter Weiterführend besitzen SVMs Hyperparameter [22], die vom Modell nicht gelernt, sondern extern festgelegt werden müssen. Der Parameter C ist so ein zentraler Regulierungsparameter einer SVM. Er steuert das Verhältnis zwischen Trainingsfehlern und der Margin-Breite (siehe Abbildung 12) und hat somit direkten Einfluss auf die Qualität des Modells:

$$min(\frac{1}{2}|w|^2 + C\sum_{i=1}\epsilon_i) \tag{44}$$

mit der Bedingung

$$y_i(w^T x_i + b) \geq 1 - \epsilon_i, \tag{45}$$

wobei $\epsilon \in [0, 1]$ als Spielraumvariable angibt, wie nah ein Punkt an der Trennebene liegt. Ist $\epsilon \geq 1$, ist der Punkt falsch klassifiziert. Kleine C-Werte erlauben eine breite Margin, was zu einer besseren Generalisierung führt, jedoch auch zu mehr Fehlklassifikationen. Das Modell wird somit mit einer weicheren Grenze reguliert. Große C-Werte hingegen sorgen für eine schmale Margin, lassen weniger Fehlklassifikationen zu, aber erhöhen auch das Risiko für Überanpassung. Eine typische Methode zum Bestimmen von C ist das sogenannte Grid Search. Dabei werden für C verschiedene, in der Regel exponentiell ansteigende, Werte eingesetzt und das trainierte Modell mit einer Cross-Validation (siehe auch Unterabschnitt 5.1.1) bewertet. Die beste Modellleistung wählt so den optimalen Parameter aus. Dieses Vorgehen wird im Zuge dieser Arbeit für die Ermittlung von C ebenfalls verwendet (siehe Unterabschnitt 5.3).

Historie In Bezug auf die Geschichte der SVM [45] ist das Konzept bereits Mitte des 19. Jahrhunderts in den 1960ern entwickelt worden. Offiziell vorgestellt wurde die SVM im Jahr 1995 und etablierte sich schnell als eine der Standardmethoden, vor allem im Bereich der Text-Klassifikation. Ursprünglich wurde die SVM für binäre Klassifikation entwickelt, in den frühen 2000ern wurde sie jedoch für die Mehrklassen-Klassifikation erweitert.

2.2.3 Merkmalsvektor

Entscheidend für die Performance des Modells ist die Qualität des Merkmalsvektors. Dies betrifft sowohl den Input für die Vorhersagen im laufenden Betrieb als auch derer, die für das Training verwendet werden. Wie die Qualität des Modells auf Basis einer verbesserten Eingabe durch Merkmals-Konstruktion (engl. feature engineering) [21] beeinflusst werden kann, wird im Folgenden vorgestellt und ist in Tabelle 2 zusammengefasst. Zunächst ist der Informationsgehalt der Merkmale entscheidend. Die enthaltenen Informationen sollten relevant für die Trennung der gewünschten Klassen sein, jedoch nicht in redundanter Form vorliegen. Es sollten verschiedene Perspektiven auf die Klasse kombiniert werden. Im semantischen Kontext sind dies beispielsweise sowohl statistische Mittel als auch Distanzmaße. Um die Klassifikation auch bei Schwankungen der Input-Werte stabil zu halten, sollten robuste Merkmale gewählt werden. Dies verhindert zusätzlich Überanpassung beim Training. Da verschiedene Merkmale verschiedene Werteskalen verwenden, müssen diese vereinheitlicht werden. Dies verhindert die Bevorzugung hoher Werte beziehungsweise Missinterpretationen. Von Klyueva [21] besonders hervorgehoben wird die Nutzung und Kombination von Merkmalen aus anderen vorangegangenen Modellen wie z.B. Random Forest.

Tabelle 2: Auflistung relevanter Charakteristiken des Aufbaus eines Merkmalsvektors für eine SVM

Eigenschaft	Beschreibung
Signifikanz	Für Trennung relevant, nicht redundant
Diversität	Kombination verschiedener Merkmals-Erhebungen
Robustheit	Stabil gegenüber Schwankungen
Skalierung	Harmonisierung verschiedener Skalen
Kombination	Verwendung von Merkmalswerten aus anderen Modellen

2.2.4 Abrufgestützte Generierung

Definition 2.6. *Die abrufgestützte Generierung [23] (engl. Retrieval-Augmented Generation (RAG)) ist ein Modellarchitekturansatz, bei dem ein KI-Sprachmodell seine Antworten nicht ausschließlich über Modellparameter generiert, sondern zusätzlich externe Quellen als Kontext miteinbezieht.*

Das Ziel besteht darin, dass die Antworten des Modells nachprüfbarer werden und spezielles Domänenwissen integriert werden kann. Der RAG-Mechanismus besteht dabei aus zwei Hauptkomponenten, dem *Retriever* und dem *Generator*. Der Retriever kodiert Dokumente und Queries als Embeddings und fungiert als Zugangsmöglichkeit zu den externen Wissensquellen. Diese werden nicht in den Parametern des Sprachmodells gespeichert und werden aus dem Grund auch

als nicht-parametrisches Gedächtnis bezeichnet. Auf Basis der textuellen Eingabe in das Sprachmodell werden die ähnlichsten thematisch passenden Dokumente durch den Retriever ermittelt. Der Generator ist entsprechend das parametrische Gedächtnis in Form eines Transformers und erzeugt den Output des Modells. Dies setzt sich aus der Eingabe, dem antrainierten und dem externen Wissen zusammen.

Eine Möglichkeit, externes Wissen abzulegen, besteht dabei in einer Vektordatenbank [14]. Diese Wissensbasis lässt sich ohne erneutes Training ersetzen oder aktualisieren, was wiederum eine Ressourceneinsparung und mehr Flexibilität der Daten bedeutet. Eine Vektordatenbank eignet sich zudem als Datengrundlage für STS, da sie semantische Suche unterstützt. Allgemein bieten Vektordatenbanken auch außerhalb der RAG eine Methode der externen Wissensquellen für ML-Modelle.

2.3 Vorstellung der betrieblichen Hintergründe

Im folgenden Unterkapitel wird näher auf die betriebliche Kontextumgebung eingegangen, welche die Grundlage für die in Unterabschnitt 1.2 vorgestellte Problemstellung bildet. Dies beinhaltet zunächst eine Einführung in die Hintergründe des Train Zero Labors, das Validierungstests im Kontext der funktionalen Sicherheit durchführt. Dabei werden die strukturellen und operativen Aspekte des Labors betrachtet, welche ein besseres Verständnis für die Rahmenbedingungen und Anforderungen der durchzuführenden Tests ermöglichen. Des Weiteren werden der Kontext und der Aufbau der funktionalen Testspezifikationen erläutert, welche die Basis der Validierungstests bilden und den Input des angestrebten Zuordnungssystems zu den Bedienelementen darstellen. Ein weiterer Bestandteil dieses Kapitels ist die Vorstellung des bestehenden Systems zur Teilautomatisierung der Validierungstests. Es wird aufgezeigt, inwieweit die Methoden des maschinellen Lernens eine Effizienzsteigerung ermöglichen und wie das Prototyp-System aufgebaut ist, welches im Zuge der Evaluierung mit dem hier erstellten Modell verglichen wird.

2.3.1 Validierungstests in der funktionalen Sicherheit

Definition 2.7. *Funktionale Sicherheit (FuSi) [9] ist ein Sicherheitsprinzip, nach dem komplexe Systeme so ausgelegt werden, dass diese auch bei auftretenden Fehlfunktionen keine Gefahr für beteiligte Menschen, Sachwerte und die Umwelt darstellen.*

Vor allem im Eisenbahnverkehr sind entsprechende Vorkehrungen von fundamentaler Relevanz, da bei einem Systemversagen eine große Anzahl an Passagieren und andere beteiligte Personen in Lebensgefahr geraten. Die Grundlage bildet dabei die internationale Norm *IEC 61508* [8], die den Fokus auf elektrische und programmierbare sicherheitsrelevante Systeme legt.

Im V-Modell sind Validierungstests am Ende des Gesamtprozesses vorgesehen, wie

in Abbildung 13 blau hervorgehoben ist. Die Spezifikationen der Anforderungen und deren Umsetzung erfolgt auf der linken Seite des V-Modells. Diese werden im Zuge zahlreicher Tests auf der rechten Seite validiert (siehe Abbildung 13). Zum Bereich der Validierung gehören dabei vor allem die Integrations- und Validierungstests, die das zugweite Systemverhalten prüfen und für die Zulassung vorbereiten. Der Unterschied zwischen Integrations- und Validierungstests wird im Folgenden kurz erläutert.

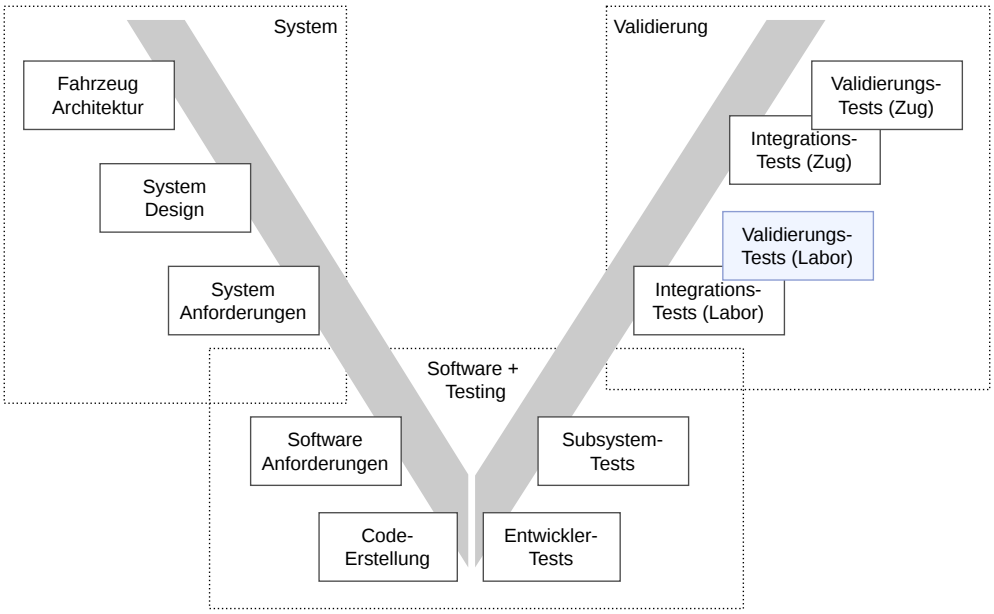

Abbildung 13: Darstellung des V-Modells mit den wichtigsten Prozessbestandteilen und Testschritten

Integrationstest Die Integrationstests prüfen das fehlerfreie funktionale Zusammenspiel der kompletten Zugsoftware, bei welcher alle Subsysteme zusammengefügt werden. Dies findet im Zusammenspiel mit der zugehörigen elektronischen und mechanischen Hardware statt.

Validierungstest Die Validierungstests demonstrieren vor allem die Erfüllung der funktionalen Anforderungen auf Architekturebene an das gesamte Zugsystem. Dies schließt das Verhalten im normalen sowie auch im Fehlerbetrieb ein und verifiziert somit auch die integrierten FuSi-Anfoderungen. Allgemein dienen Validierungstests als eine formale Dokumentation der vertraglichen Erfüllung der Kundenanforderungen.

Treten während des Test-Prozesses Fehler auf, werden diese zur Behebung an die zuständige Abteilung zurückgegeben, was im späteren Verlauf eine erneute Validierung zur Folge hat. Da sich die Prozessbestandteile und das validierende Gegenstück im V-Modell gegenüber liegen, zieht eine Fehlerbehebung zu einem so späten Zeitpunkt im V-Modell einen entsprechend längeren Korrektur-Prozess mit sich. Dies ist exemplarisch in Abbildung 14 gezeigt. Je früher im Testing-Prozess ein Fehler aufgedeckt wird (z.B. bereits während der Softwaretests), desto eher lässt sich dieser beheben noch bevor weitere Tests durchgeführt werden.

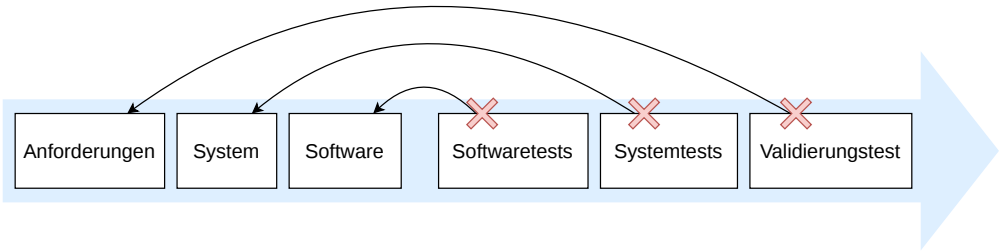

Abbildung 14: Veranschaulichung der Verzögerungsausmaße bei fortgeschrittenem Prozess

Entsprechend ist es wichtig, Fehler frühzeitig zu erkennen und zu beheben. Aus diesem Grund werden die Validierungstests zunächst in dem sogenannten *Train Zero Labor* durchgeführt, bevor die Tests auf einer realen Lok stattfinden. Dies hat zeitliche und finanzielle Vorteile und reduziert die Test-Korrektur-Iterationen auf dem Fahrzeug.

2.3.2 Train Zero Labor

Das *Train Zero* Labor ist eine Zwischenstation der funktionalen Tests auf Systemebene im Testlabor und den Test auf der realen Lok. Es handelt sich um ein Labor, welches neben dem Equipment für Softwaretests auch mit sämtlicher Kleinelektronik ausgestattet ist, die für Tests auf einer realen Lok benötigt wird. Dies schließt beispielsweise einen vollausgestatteten Führertisch (FT) mit ein sowie MIOs, Relais und die Geräte sämtlicher System- und Sicherheitskomponenten. Hardware, deren Anschaffung ein Kosten-Nutzen-Verhältnis in Bezug auf Validierungstests nicht rechtfertigt (beispielsweise der Motor, der Stromabnehmer oder Hydraulikkomponenten), werden über eine Simulation mit den Hardwarekomponenten verbunden. Dies ermöglicht funktionale Tests, die einen Großteil der realen Elektronik inklusive tatsächlicher Bedienhandlungen am FT beinhaltet. So lassen sich im Vorfeld zulassungsrelevante Tests durchführen, ohne kostenintensive Ressourcen und Personal für Fahrzeugtests investieren zu müssen. Wenn die Tests am Ende des Prozesses auf der Lok durchgeführt werden, ist die Wahrscheinlichkeit des Entdeckens neuer Fehler erheblich vermindert.

Da diese Arbeit mit sicherheitsrelevanten Testspezifikationen und in diesem Zusammenhang mit der Bedienung von Hardwareelementen des FTs arbeitet, wird dieser kurz vorgestellt. So wie der FT in Abbildung 15 als Bestandteil einer Lok abgebildet ist, wird dieser auch als Modell mit allen realen Bedienelemente im Train Zero verwendet. Operationen wie das Heben des Pantographen, das Schließen des Hauptschalters oder das Beschleunigen über den Fahr-Brems-Hebel lassen sich so für die Validierungstests bereits mit realer Hardware durchführen. Die verbundene Simulation reagiert entsprechend auf die Bedienhandlungen.

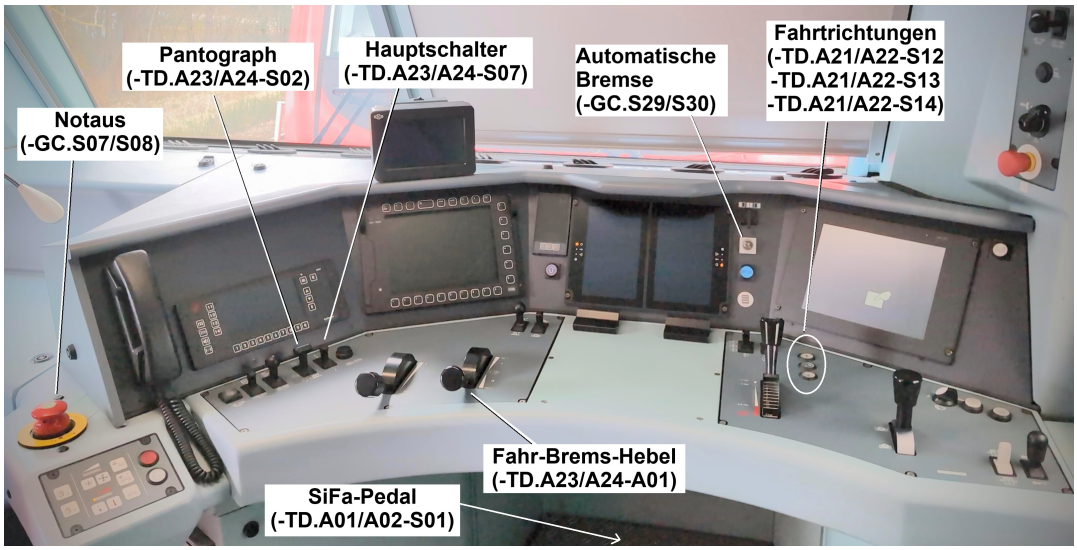

Abbildung 15: Führertisch einer TRAXX AC3 Lok inklusive Kennzeichnung relevanter Bedienelemente für diese Arbeit; Foto: Thomas Woge

2.3.3 Funktionale Test-Spezifikationen

Funktionale Test-Spezifikationen (FTS) sind detaillierte Testanweisungen zu bestimmten Funktionalitäten eines Schienenfahrzeugs und dienen als Handlungsvorschriften im Zuge der Validierungstests. Sie beschreiben die zu testenden Funktionen des Fahrzeugs, mit welchen Konfigurationen, Vorbedingungen und in welcher Reihenfolge die Tests durchgeführt werden sowie welche Ergebnisse aus den Tests erwartet werden. Sie verkörpern somit die Verbindung zwischen Systemanforderungen (engl. functional requirement specification (FRS)) und der Fahrzeugvalidierung und weisen die Erfüllung der jeweiligen Funktionen nach. Eine wichtige Rolle spielen FTS als Nachweisdokumente für spätere Zulassungen oder Zertifizierungen.

2.3.4 Bestehender Prototyp zur Teilautomatisierung von Validierungstests

Um die Durchführung von Validierungstests einfacher und ressourcenschonender zu gestalten, besteht die Option, diese zu automatisieren. Vor allem, da die Tests bei jeder neuen Softwareversion durchgeführt werden müssen, ist eine Automatisierung sinnvoll und ist im Bereich der Softwaretests bereits gängige Praxis. Für Validierungstests besteht die Herausforderung, die jeweiligen Hardwareinteraktionen in die Automatisierung einzubinden. In einer vergangenen Masterarbeit [19] wurde dieses Problem aufgegriffen und in Form eines teilautomatisierten prototypischen Systems umgesetzt. Dieses erzeugt aus eingelesenen FTS-Daten spezielle Sequenzdateien, welche wiederum von einer Testsoftware des Train Zero Labors zur Durchführung von FuSi-Tests genutzt werden.

Das Programm benötigt ebenfalls eine Zuordnung der Testschritte zu deren zugehörigen Bedienelementen, um das entsprechende Schnittstellensignal zur Software zu identifizieren. Dies wird im Rahmen des Prototypen über einen Fuzzy-Stringvergleich [32] realisiert, welcher methodisch auf der Levenshtein-Distanz [40] basiert. Es wird angestrebt, im Rahmen dieser Arbeit eine effiziente Alternative zu der genutzten Methode zu erarbeiten und zu integrieren.

2.4 Zusammenfassung

Zurückblickend gibt dieses Kapitel einen umfassenden Überblick über die theoretischen Grundlagen, die für die Entwicklung eines ML-gestütztes Zuordnungssystem relevant sind. Zunächst wurden die Methoden der semantischen Textrepräsentation vorgestellt, darunter grundlegende ML-Konzepte, Verfahren des überwachten Lernens sowie elementare Bausteine der NLP-Pipeline. Ein besonderes Augenmerk lag auf der semantischen Textähnlichkeit und der Erzeugung kontextsensitiver Embeddings, da diese die Grundlage der späteren Klassifikation bilden. Darauf aufbauend folgte eine vertiefte Darstellung der Sentence-Transformer und das in dieser Arbeit verwendete Modell *all-MiniLM-L6-v2*. Die Eignung des Modells wurde mit der effizienten Inferenzzeit, Ressourcennutzung und der guten semantischen Qualität begründet. Im Anschluss wurden verschiedene Klassifikationsmethoden erläutert, insbesondere die Support Vector Machine, deren Struktur, Kernel-Mechanismus und Parameter eine zentrale Rolle im späteren System einnehmen. Abschließend wurde auf den betrieblichen Kontext eingegangen. Dazu gehören die funktionale Sicherheit im Eisenbahnbereich, die Bedeutung des Train Zero Labors als Testumgebung sowie der Aufbau und die Rolle funktionaler Testspezifikationen. Außerdem erfolgte eine Einordnung des bestehenden Prototypen zur teilautomatisierten Testsequenzerstellung, dessen bisherigen Heuristiken die Grundlage und Motivation dieser Arbeit bilden.

3 Modellierung

Die folgenden Kapitel stellen den Hauptteil der Arbeit dar und fügen die in Abschnitt 2 vorgestellten Grundlagen zu einem Modell-Entwurf zusammen. Das Ziel dieses Kapitels ist die Planung und Modellierung einer Komposition aus dem Sequenz-Transformer *all-MiniLM-L6-v2* im Zusammenspiel mit den Methoden der Zuordnung nicht-semantischer Bedienelemente. Diese Methoden werden dabei in einem konkreten Anwendungskontext veranschaulicht und die einzelnen Schritte von der Datenvorbereitung bis zur Implementierung systematisch nachvollziehbar dargestellt. Das Hauptkapitel gliedert sich dabei in die Modellierung der Zuordnungsmechanismen, welche aus einer Komposition aus Vektordatenbank zur Kontextunterstützung und einer SVM bestehen. Diese werden in eigenen Unterkapiteln behandelt. Zudem wird das modellierte Gesamtsystem in den Kontext der teilautomatisierten Validierungstests gesetzt und in einer modularen Architektur vorgestellt. Die entsprechende Integration wird im nachfolgenden Hauptkapitel Abschnitt 4 kurz vorgestellt, um eine vollständige Dokumentation zu gewährleisten.

3.1 Modellierung der Zuordnungsmöglichkeiten nicht-semantischer Bezeichner

Der nachfolgende Abschnitt dokumentiert die Architektur und konkrete Bestandteile des Zuordnungssystems. Zunächst erfolgt eine Übersicht über den Aufbau der Vektordatenbank, welche als Basis der semantischen Ähnlichkeitssuche eine wichtige Rolle spielt. Ergänzt wird der Abschnitt durch eine Betrachtung der sich ergebenden Herausforderungen im Hinblick auf die Handhabung der doppelten Fahrkabinen. Im weiteren Verlauf wird die konkrete Zusammensetzung des Merkmalsvektors unter Berücksichtigung seiner qualitativen Eckpunkte betrachtet. Der ganzheitliche Ablauf des Zuordnungsprozesses wird zusammen mit der Trainingsstrategie zum Schluss vorgestellt.

3.1.1 Architektur

Bei einer Kombination von Sentence-Transformer mit einer Klassifizierung ist eine hybride Architektur notwendig, welche in Abbildung 16 abgebildet ist. In dieser wird die MiniLM zur Erzeugung der Vektordatenbank und des Merkmalsvektors verwendet. Die Erstellung des Merkmalsvektors wird zusätzlich durch eine STS gestützt, welche durch die Vektordatenbank und somit indirekt mithilfe des Sentence-Transformers durchgeführt wird. Der Merkmalsvektor wird anschließend einem Klassifikator übergeben, welcher durch eine SVM gestellt wird.

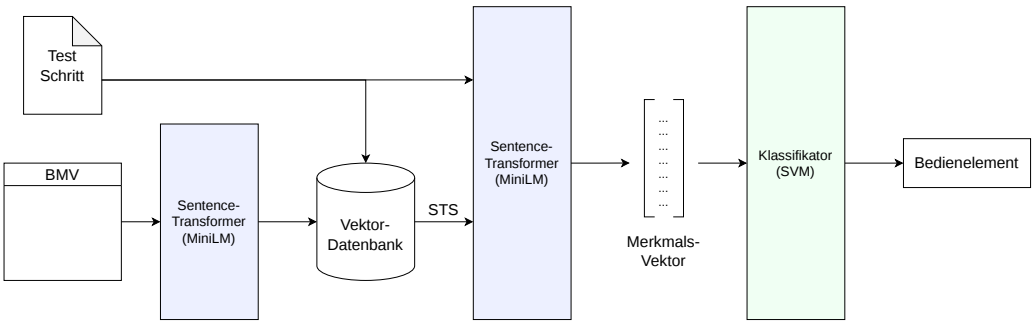

Abbildung 16: Darstellung der hybriden Architektur mit Sentence-Transformer und SVM-Klassifikator als Hauptbestandteile

Im Gegensatz zu alternativen ML-Modellen, welche eine kombinierte Embedding-Erzeugung mit einer Klassifikation vereinen, ist der hybride Ansatz durch eine erhöhte Flexibilität und Effizienz bei kleinen Datensätzen gezeichnet. Die semantische Repräsentation und Klassifikator sind getrennt, was einen leichteren Austausch derselben und ein getrenntes Training begünstigt. Dies spart Zeit und Ressourcen, da das Sprachmodell nicht neu angepasst werden muss. Bei der hybriden Methode lassen sich Embeddings gesondert berechnen und beispielsweise auch für die Erstellung der Vektordatenbank verwenden, während reine Klassifikationsmodelle in der Regel nur für die Klassifikation verwendet werden können. Zudem sind vortrainierte Embedding-Modelle auf große Textmengen optimiert und liefern stabile semantische Repräsentationen. Diese sind entscheidend für den Erfolg der Klassifikation. Zusammenfassend verwendet die hybride Methode zwei Modelle (Sentence-Transformer und Klassifikator), die auf die jeweilige Aufgabe spezialisiert sind. Ein kombiniertes Modell spart im Gegensatz zu einem Hybridmodell Code und Strukturierung, jedoch gibt es Abstriche in Bezug auf Flexibilität und Ressourcenverwendung, worauf in dieser Arbeit großer Wert gelegt wird.

3.1.2 Embedding-Datenbank

Im Gegensatz zu Vektor- beziehungsweise Embedding-Datenbanken [14], deren Inhalt aus zerlegten Dokumenten oder Websites (Chunks) erstellt werden, setzt sich die hier verwendete Datenbank aus atomar, entitätsbasierten Einträgen zusammen. Diese werden als eigenständige Dokumente abgelegt. Da es sich um eine Ablageform des BMVs handelt, repräsentiert jedes Dokument ein konkretes Bedienelement. Ein Element entspricht dabei einer Zeile des BMV (siehe Abbildung 17). Als Informationen ist dabei neben der nicht-semantischen Bezeichnung die Beschreibung des Elements enthalten sowie die Positionsinformationen. Diese sind besonders relevant, da jedes Element durch die zwei Führerstände in doppelter Ausführung vorhanden ist, aber jeweils unterschiedliche Bezeichner besitzt.

	A	B	C	D	E
1	POSNR	BENENNUNG1	BENENNUNG2	CAB	LOCATION
66	-GC.S02	pushbutton	Activation of the cab	Cab 2	+85
67	-GC.S03	illuminated pushbutton	Deactivation of the cab	Cab 1	+15
68	-GC.S04	illuminated pushbutton	Deactivation of the cab	Cab 2	+85
69	-GC.S07	emergency brake push button	emergency brake push button	Cab 1	+11
70	-GC.S08	emergency brake push button	emergency brake push button	Cab 2	+81
71	-GC.S10	rotary switch	Isolation switch traction safe	Cab 1	+51

Abbildung 17: Exemplarischer Auszug aus dem BMV

Aufbau der Dokumente Der Aufbau eines Dokuments, welches als Basis für einen Eintrag der Vektordatenbank dient, ist in Abbildung 18 dargestellt. Die textuellen Beschreibungen der Elemente bilden den Haupteintrag. Die Informationen, wie der Bezeichner (id) und die Lokalisierung (cab, location_num) werden in Form von Metadaten gesichert. Der Haupteintrag selbst besteht aus sechs beschreibenden Sätzen des Bedienelements, wie sie teilweise bereits im BMV vorhanden sind oder in der FTS vorkommen können. So sollen bereits syntaktische Variationen abgedeckt werden, was später die STS vereinfacht.

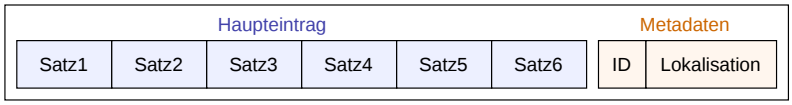

Abbildung 18: Inhaltliche Aufteilung eines Dokuments in der Vektordatenbank

Informationsabruf Analog zu der Vorgehensweise im RAG-Kontext erfolgt der Abgleich eines Eingabetextes mit dem Datenbankinhalt durch semantische Ähnlichkeit der Embeddingrepräsentationen. Dabei wird von der Datenbank die Kosinus-Distanz [27] verwendet. Die Top-k relevanten Dokumente werden anschließend zurückgegeben, einschließlich eines Ähnlichkeits-Scores, welcher dem Ranking der durchgeführten STS entspricht.

Herausforderungen Die bereits angesprochene Doppelungen der Bedienelemente im Führerstand stellen bei der Verwendung der STS und Klassifizierung durch die SVM eine zentrale Herausforderung dar. Die Bezeichner sind unterschiedlich, aber die individuellen Beschreibungen der Einträge sind nahezu identisch. Sie unterscheiden sich nur in der Lokalisation. Diese feine Unterscheidung führt zu Unsicherheiten bei der Zuordnung. Es besteht die Möglichkeit, die Information der Lokalisation eine größere Gewichtung zu verleihen. Dabei wird die Wahrscheinlichkeit einer korrekten Zuordnung erhöht, jedoch bleibt ein Restrisiko einer falschen Zuordnung. Um diese Problematik vorgreifend zu vermeiden, besteht die Möglichkeit, getrennte Vektordatenbanken zu verwenden. Da der besetzte Führerstand für einen Testschritt bekannt sein muss, lässt sich somit die zugehörige Datenbank für die STS im Voraus auswählen. Die Unsicherheit bezüglich der korrekten Auswahl des Führerstandes entfällt somit und erhöht die Sicherheit in der Zuordnung.

3.1.3 Merkmalszusammensetzung

Als Methode zur Ermittlung des tatsächlichen zugehörigen Bedienelements zum Testschritt wird die Klassifikationsmethode mittels SVM festgelegt. Wie bereits in Unterabschnitt 2.2.2 ausgeführt, ist die Qualität des Merkmalsvektors entscheidend für den Erfolg des Modells. Wichtige Eigenschaften sind dabei die Signifikanz der enthaltenen Informationen, die Diversität der Informationsquellen, die Robustheit bei Schwankungen, eine angepasste Skalierung aller Merkmale und gegebenenfalls die Integration von Merkmalen, die bereits aus anderen Modellen gewonnen wurden. Die Eigenschaften sind in Tabelle 3 zusammen mit solchen Merkmalen aufgeführt, welche die entsprechenden Eigenschaften erfüllen. Im Folgenden werden die entscheidenden Punkte bezüglich der Merkmalsauswahl und der Zusammensetzung erörtert und die möglichen Strukturen des Inputvektors vorgestellt.
Zunächst ist die Nutzung zweier semantischer Maße geplant. Dies beinhaltet zum Einen eine STS, welche ein Top-k-Ranking und die zugehörigen STS-Scores wiedergibt und zum Anderen die Kosinus-Distanz der Texte.

Tabelle 3: Auflistung der potentiellen Bestandteile zur Erstellung eines qualitativ ausgewogenen Merkmalsvektors

Eigenschaft	Merkmal
Signifikanz	STS-Score (Semantik), Aggregation, Verhältnisse, Input-Embedding, Top-k Embedding
Redundanz	Verwendung jeweils eines signifikanten Merkmals
Diversität	STS-Score, Kosinus-Distanz, Aggregation, Top-k Embedding
Robustheit	Aggregation, Verhältnisse, Top-k Embedding
Skalierung	Normalisierung des Merkmalsvektors vor dem Training/der Vorhersage

STS-Score Der STS-Score ist ein Maß für die Ähnlichkeit zwischen der textuellen Eingabe und den Einträgen der Vektordatenbank. Die semantische Ähnlichkeit wird dabei berücksichtigt, wodurch der STS-Score eine hohe Relevanz für eine Klassenzugehörigkeit liefert. Die Information besitzt aufgrund der direkten semantischen Relation eine hohe Signifikanz und sind Bestandteil eines ausgewogenen Merkmalsvektors.

Kosinus-Distanz Die Kosinus-Distanz misst die geometrische Nähe im Vektorraum und vergleicht die semantische Ähnlichkeit damit nur indirekt. Sie bietet eine andere Perspektive auf die selbe Struktur und würde somit zur Diversität beitragen. Da sowohl der STS-Score als auch die Kosinus-Distanz ein Maß für semantische Ähnlichkeit sind, besteht hier die Gefahr der Redundanz. Jedoch werden die Informationen mit unterschiedlichen Hintergründen erworben. Die STS stellt einen Vergleich mit den Einträgen aus der Vektordatenbank dar, welcher auf einem komplexeren Prozess basiert als die reine Betrachtung der Distanz im Vektorraum. Der Einfluss beider Maße auf die Qualität des Modells in Bezug auf Redundanz wird im Zuge der Evaluierung in Abschnitt 5 untersucht.

Aggregation Aus den Ähnlichkeits-Metriken, allen voran dem STS-Score, lassen sich wiederum aggregierte Merkmale ergänzen. Diese stellen jeweils den Mittelwert und die Varianz der erhobenen Scores dar. Zusätzlich werden Informationen wie der Maximal- und Minimalwert der STS-Scores eingebracht, aus welchen wiederum der Werteumfang (engl. range) ermittelt werden kann. Diese Informationen geben Hinweise auf die Struktur der Top-k Treffer. Eine hohe Varianz deutet beispielsweise auf eine heterogene Nachbarschaft hin und ein hoher Maximalwert auf einen stark ähnlichen Treffer. Die Range gibt Hinweise auf die Spannweite der Ähnlichkeiten, ob beispielsweise alle k Treffer eine hohe Ähnlichkeit besitzen oder ein starkes Gefälle zwischen dem besten und schlechtesten Treffer besteht. Diese Informationen lassen sich den Bereichen der Robustheit zuordnen, da Ausreißer geglättet

werden. Zudem liefern sie gute globale Strukturinformationen, was der Diversität des Merkmalsvektors zugute kommt. Vor allem der Mittel- und Maximalwert liefern signifikante Informationen.

Verhältnisse Abgesehen von den erhobenen Statistiken im Zuge der STS stellen relative Merkmale eine weitere wertvolle Information dar. Vor allem das Verhältnis der Scores der beiden besten Treffer der STS. Diese liefern eine stabilisierte Bewertung der Stärke des Top-Treffers. Hat dieser einen deutlich besseren Score als die anderen, kann der Treffer mit einer hohen Wahrscheinlichkeit als korrekt angesehen werden. Bei einer kleineren Differenz zum nächstbesten Score, ist die Qualität der Klassifizierung entscheidend, damit der korrekte Treffer identifiziert wird. Dieses Maß ist signifikant in Bezug auf die Stärke des besten STS-Treffers und aufgrund der relativen Angabe weniger anfällig für Skalenschwankungen. Zur Vermeidung von Redundanzen wird nur das Verhältnis der ersten beiden Treffer verwendet, da dieses am aussagekräftigsten ist.

Input-Embedding Das Input-Embedding in Form einer Vektorrepräsentation beinhaltet den ungekürzten semantischen Kontext des Eingabetextes und stellt das stärkste Einzelmerkmal für die Klassifikation dar. Es ist bereits in Vektorform vorhanden, was eine Integration in den Merkmalsvektor erleichtert. Als Hauptinformationsquelle ist es in den Bereich der Signifikanz und Skalierung einzuordnen, da er unabhängig der Textlänge immer die selbe Dimension besitzt.

Gewichtetes Top-k Embedding Die gewichtete Summe der Top-k Embeddings aus der STS integriert das externe Wissen der Vektordatenbank in die SVM-Eingabe. Da mehrere in Frage kommende Klassen vorgefiltert werden, erfolgt durch die gewichtete Summe eine Glättung von Ausreißern und eine Erhöhung der Robustheit. Das Ranking der Ergebnistreffer wird dabei durch Einbeziehung des STS-Scores als Gewichtung berücksichtigt. Es erweitert zudem das Input-Embedding um die Top-k Treffer der STS und trägt somit zur Diversität bei.

Zusammenfassend bilden die Merkmale insgesamt die Semantik, die Nachbarschafts-Struktur, die Statistik und den Kontext ab. Dabei sind sie robust gegenüber Rauschen und vereinzelten Schwankungen. Sie bilden einen numerischen Vektor, welcher als Input für eine SVM gut geeignet ist. Das Wissen kommt dabei sowohl aus den Input-Informationen als auch aus einer externen Datenquelle, um den Kontext der Eingabe im Vorfeld zu spezifizieren. Der Zusammenhang aller Merkmals-Typen mit den qualitativen Eigenschaften ist in Tabelle 3 und Tabelle 4 gegeben. Während der Evaluierung werden verschiedene Zusammensetzungen des Merkmalsvektors getestet. Aus diesem Grund ergibt sich der schlussendliche Aufbau im Anschluss an die entsprechenden Analysen.

Tabelle 4: Auflistung der Merkmale des Merkmalsvektors für eine Klassifikation von Bedienelementen

Merkmal	Beschreibung
STS-Score	Semantische Ähnlichkeit im Kontext der STS
Kosinus-Distanz	Geometrische Nähe von Embeddings im Vektorraum
Mittelwert	Aggregation - Score der Nachbarschaft
Varianz	Aggregation - Score-Verteilung in der Nachbarschaft
Max STS	Aggregation - Stärke der Ähnlichkeit des besten Top-k Treffers
Min STS	Aggregation - Stärke der Ähnlichkeit des schlechtesten Top-k Treffers
Range STS	Aggregation - Spannweite der Ähnlichkeit in der Nachbarschaft
STS Verhältnis	Verhältnis - Stärke der STS zwischen den beiden besten Treffern
Input-Embedding	Embedding des Input-Textes
Gew.-Embedding	Kontexteinbeziehung der Nachbarschaft

3.1.4 Ablaufmodell

Der Zuordnungsablauf ist im Folgenden in Abbildung 19 dargestellt. Zunächst erfolgt die Übergabe des Testschrittes vom Hauptprogramm über die Interfaceschnittstelle. Diese übermittelt auch die Information, welcher Führerstand betroffen ist. Auf Basis dieser Information wird die entsprechende Vektordatenbank und das trainierte SVM-Modell geladen. Im Anschluss erfolgt die STS-Berechnung, indem der Testschritt der Vektordatenbank übergeben wird, welche die Top-k ähnlichsten Einträge zurück gibt. Mit diesen Informationen wird der STS-Score und die Kosinus-Distanz berechnet, welche als Bestandteile des Merkmalsvektors verwendet werden. Dieser wird in die SVM zur Vorhersage der passenden Bedienelement-Klasse eingespeist. Die SVM gibt die ermittelte Klasse sowie einen Konfidenz-Wert zurück. Basierend auf seinem Wert wird entschieden, ob dem Ergebnis in seiner Richtigkeit vertraut wird oder ob ein Human-in-the-Loop eine Verifizierung durchführen muss. Wird das Ergebnis bestätigt, wird das Programm fortgesetzt. Erfolgt eine Korrektureingabe, wird diese verwendet und zusätzlich als Feedback für eine anschließende Anpassung des Modells gespeichert.

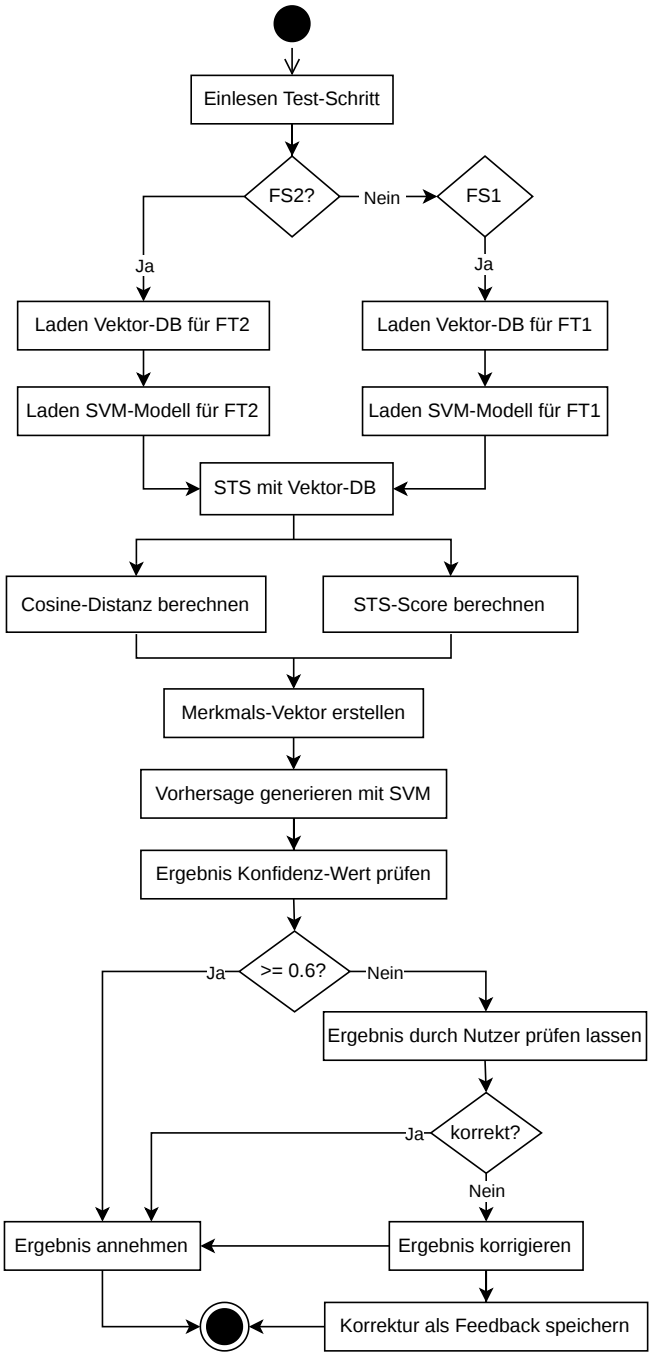

Abbildung 19: Aktivitätsdiagramm der Bedienelementklassifizierung in der normalen Nutzung; FS = FT

3.1.5 Trainingsmethode

Das Training der SVM gestaltet sich zu Beginn ähnlich wie der normale Klassifikationsablauf und ist in Abbildung 20 dargestellt. Bei diesem werden die Vektordatenbanken und weitere Ressourcen in Abhängigkeit des angegebenen Führerstandes ausgewählt. Für das Training erfolgt der selbe Prozess entsprechend für beide Führerstände gleichermaßen. In der Abbildung 20 ist der jeweilige Führerstand (engl. cab) mit *FSx* angegeben. Für jeden Führerstand wird zunächst die zugehörige BMV eingelesen. Diese kann manuell als getrennte Datei abgespeichert sein oder aber es wird nach den Einbauorten gefiltert. Aus dem BMV wird direkt im Anschluss die jeweilige Vektordatenbank erstellt. Für das Training der SVM muss zunächst der Trainingsdatensatz aufgebaut werden. Zu diesem Zweck besteht bereits ein Datensatz mit gelabelten Testschritten, diese müssen jedoch zunächst in einen Merkmalsvektor überführt werden. Zu diesem Zweck wird die Excel-Liste der gelabelten Daten durchlaufen, wobei für jeden Eintrag das STS-Ergebnis inklusive STS-Score und zusätzlich die Kosinus-Distanz berechnet wird. Aus diesen Daten wird der Merkmalsvektor aufgebaut und in einer entsprechenden Liste x gespeichert. Parallel dazu wird das Label in einer separaten Liste y abgelegt. Nachdem alle Trainingsdaten vorbereitet sind, wird der erzeugte Datensatz in Trainings- und Testdaten aufgeteilt. Das Verhältnis liegt hier bei den üblichen 80 % Trainingsdaten und 20 % Testdaten. Mit den Testdaten wird im Anschluss an das Training eine Evaluierung zur Beurteilung der Modellqualität durchgeführt.

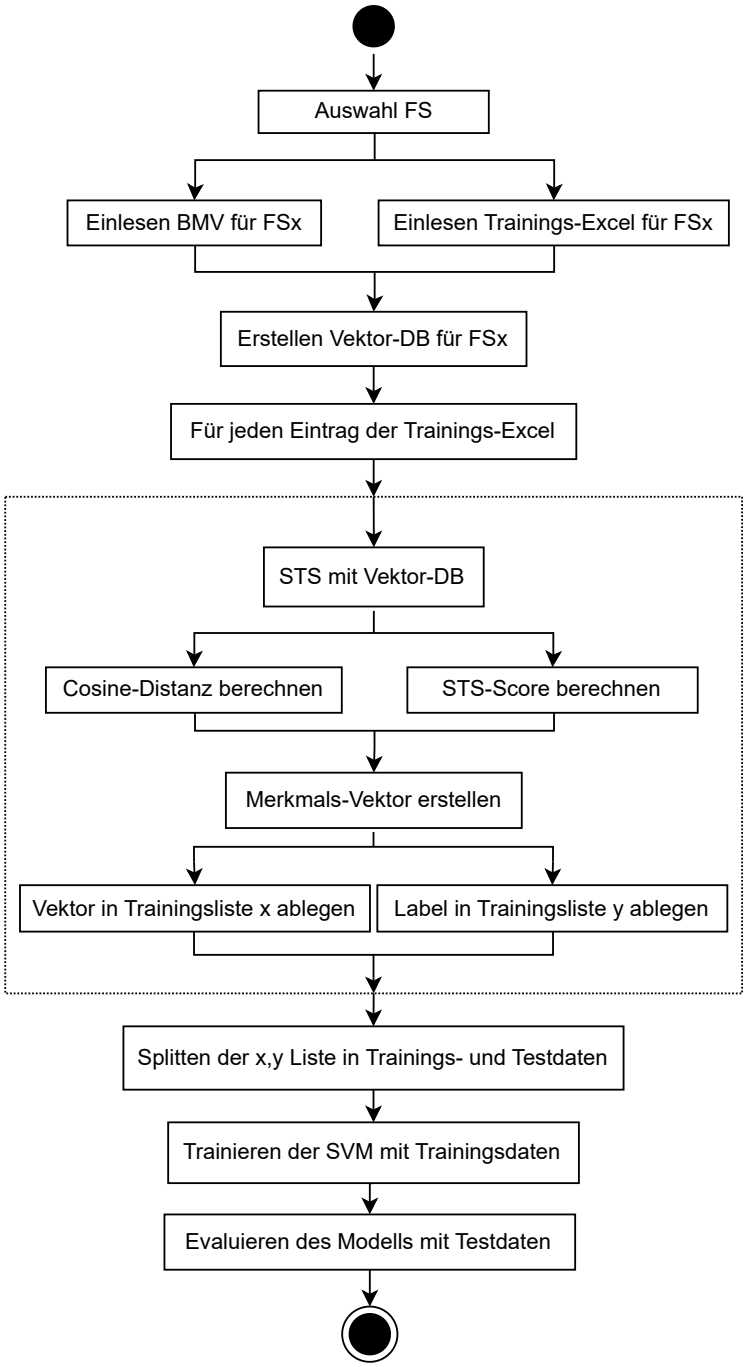

Abbildung 20: Aktivitätsdiagramm des Trainings der Bedienelementklassifizierung

3.2 Modellierung eines Systems zur Zuordnung nicht-semantischer Bezeichner

Im Folgenden wird die Modellierung des Zuordnungssystems selbst grob umrissen. Zunächst wird der Systemkontext vorgestellt, in welchen die Zuordnung eingebettet wird. Da es sich bei dem Zuordnungssystem um einen Bestandteil eines anderen prototypischen Systems handelt, ist eine effiziente Modellierung und Handhabung externer Daten relevant. Dies schließt auch entsprechend Schnittstellen ein, die im Kontext des Interfaces betrachtet werden.

3.2.1 Informations- und Kontextmodelle

Im Folgenden wird das Kontextmodell vorgestellt. Dieses beinhaltet alle externen Komponenten, mit denen das Zuordnungssystem interagiert sowie die Verbindung zum bestehenden Prototypen. Abgebildet ist das Modell in Abbildung 21. Im Anschluss folgt das Informationsfluss-Modell in Abbildung 22, in welchem die Beziehungen der Komponenten zueinander gemäß ihres Datenaustausches abgebildet werden.

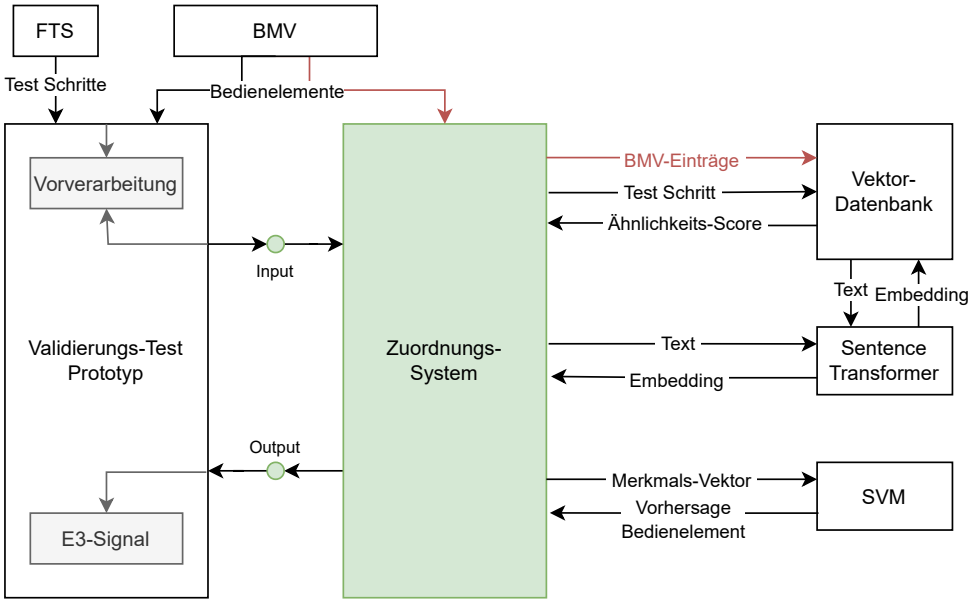

Abbildung 21: Kontextmodell der Bedienelement-Klassifizierung mit externen Komponenten und dem verbundenen Validierungstest-Prototypen. grün: Hauptsystem, rot: nur für Training verwendet

Das Hauptsystem, welches die Logik für die Zuordnung der Bedienelemente beinhaltet, wird in Abbildung 21 als Blackbox dargestellt. Relevant sind alle Kommunikationswege zu externen Informationsquellen.

Validierungstest-Prototyp Begonnen bei dem Kontext-System, dem Validierungstest-Prototypen, bezieht dieses die FTS-Daten aus einer externen Quelle und bereitet die einzelnen FTS Schritte für die Integration in das automatisierte Testprogramm vor.

Zuordnungssystem Nach der Aufbereitung greift das Zuordnungssystem ein und übernimmt den jeweiligen Testschritt und weitere Informationen wie den Besetzungsstatus der Kabinen über ein Input-Interface.

Vektordatenbank Das BMV dient für den weiteren Verlauf als Quelle für die Erzeugung der Vektordatenbank. Der genaue Aufbau ist im nachfolgenden Unterabschnitt 3.1.2 beschrieben. Das Zuordnungssystem greift im Verlauf der Zuordnung auf die erstellte Vektordatenbank zu. Dabei dient der jeweilige Testschritt als Eingabe einer STS, welche ein entsprechendes Ranking inklusive der STS-Scores zurückgibt.

SVM Der Merkmalsvektor, der unter anderen mit den STS-Scores gebildet wird, dient wiederum als Eingabe für die SVM. Diese erstellt eine Vorhersage des Bedienelements und gibt dieses über ein Output-Interface an das Zuordnungssystem zurück.

Sentence-Transformer Die Erstellung der Vektordatenbank und die Einträge des Merkmalsvektors werden durch das Transformer-Modell ermöglicht, welches die eingespeisten Texte in eine vektorielle Semantik-Repräsentation umwandelt.

In der Abbildung 22 ist der Informationsfluss abgebildet. Die Verbindungen ähneln den Kontext-Relationen in Abbildung 21, stellen jedoch die Beziehungen auf der Ebene des Informationsaustausches dar. Das Zuordnungssystem bezieht die Testschritte und die BMV über einen Import aus externen Datenquellen. Die Erstellung der Vektordatenbank erfolgt einmalig lokal für ein Projekt, jedoch können die enthaltenen Informationen im laufenden Betrieb aktualisiert werden. Während der normalen Nutzung werden STS-Anfragen an die Vektordatenbank gestellt und die Ergebnisse werden zurück an das Zuordnungssystem kommuniziert. Sowohl die Vektordatenbank als auch das Zuordnungssystem selbst nutzen den Sentence-Transformer zur Erstellung von Embeddings. Die SVM wird von dem System genutzt, indem sie trainiert, gespeichert und bei Bedarf wieder geladen wird.

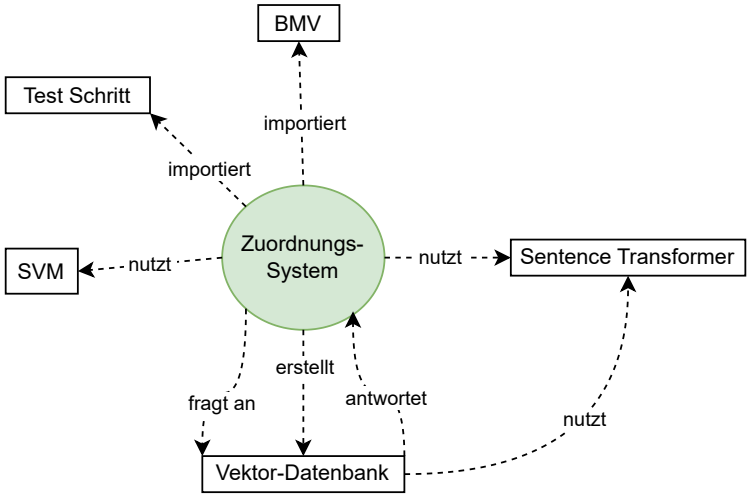

Abbildung 22: Informationsmodell der Bedienelement-Klassifizierung mit kommunizierenden Komponenten. Das Hauptsystem ist in hellem Grün dargestellt

3.2.2 Interface

Das Interface zum bestehenden Prototypen ist die einzige Kommunikationsmöglichkeit in das Zuordnungssystem hinein und aus dem System heraus. Es müssen somit alle relevanten Informationen übertragen werden können. Neben der Übergabe des Testschrittes, dem Besetzungszustand der Kabine und dem Klassifikationsergebnis müssen die ausgeführten Funktionen geregelt werden können. Beispielsweise soll das selbe Projekt sowohl die SVM trainieren als auch mit einem trainierten Modell Vorhersagen erzeugen können. Eine Möglichkeit ein Projekt über einige wenige Parameter zu regeln besteht in der Verwendung von Konfigurationen. Ein einziger Parameter fungiert dabei als ein Schalter, welcher gezielt Codeabschnitte über bestimmte Parameterwerte an- und ausschaltet. Im Code selbst wird dies durch einfache If-Else-Abfragen realisiert. Das Zuordnungssystem ist somit in seiner Wirkungsweise sowohl durch einen Aufruf aus einem externen Programm als auch durch einen einfachen Kommandozeilenbefehl über die Interfaceparameter steuerbar.

3.3 Zusammenfassung

In diesem Kapitel wurde das ML-basierte Zuordnungssystem konzeptionell modelliert. Im Zuge dessen erfolgte der Entwurf einer hybriden Architektur, welche die Sentence-Transformer-Embeddings mit einer SVM-Klassifikation kombiniert. Der erste Schwerpunkt lag zunächst auf der Struktur der Vektordatenbank, die aus atomaren BMV-Einträgen aufgebaut wird und semantische Kontextinformationen für eine spätere STS-Berechnung bereitstellt. Weiterführend umfasste die Modellierung eine detaillierte Konstruktion des Merkmalsvektors und dessen in Betracht kommende Bestandteile. Diese kombinieren das Testschritt-Embedding mit statischen und relationalen Merkmalen der STS, einschließlich aggregierter Scores. Im Zusammenhang mit dieser Auswahl wurde begründet, warum die gewählte Kombination die Robustheit, Diversität und Signifikanz der Merkmalsrepräsentation steigert. Im Anschluss daran wurde der Ablauf des gesamten Klassifikationsprozesses von der Entgegennahme des Testschrittes über die STS bis zur SVM-Klassifikation beschrieben. Mechanismen wie Konfidenzbewertung und Human-in-the-Loop sind dabei integrale Bestandteile, um Anforderungen der funktionalen Sicherheit zu gewährleisten. Das Kapitel schließt mit einer Einordnung des Modells in das Gesamtsystem auf Architekturebene. Inbegriffen waren dabei ein Kontext-, Informationsfluss- und Schnittstellenmodell. Diese stellen sicher, dass die ML-Komponente modular in den bestehenden Testsequenz-Prototypen integriert werden kann und bildet die Grundlage für die anschließende Implementierung.

4 Integration

Das folgende Kapitel beleuchtet die praktische, prototypische Umsetzung der Modellierung in die bestehende Toolchain. Dies ist ein Teil der Arbeit, welcher nicht den Hauptbestandteil einnehmen soll, jedoch einen technischen Beitrag zur Lösung der Problemstellung beiträgt. Aus diesem Grund wird dieses Kapitel einen geringeren Umfang einnehmen als die Erörterung der Methodik, erfüllt jedoch einen wichtigen Teil für die wissenschaftliche Dokumentation der Umsetzung. Sie befasst sich im Folgenden mit der Integration des erstellten Modells in die bereits bestehende Software-Umgebung. Zudem wird die Umsetzung der eigenen Anwendung zur FTS-Zuordnung zu den Bedienelementen umrissen und ein Überblick über auftretende Herausforderungen gegeben.

4.1 Technische Umsetzung eines Systems zur Zuordnung nicht-semantischer Bezeichner

Die Umsetzung des Systems zur Zuordnung der Bedienelemente wird in einer Python-Umgebung [1] durchgeführt. Das System enthält eine strukturierte Hauptanwendung, die auf Ressourcen beiliegender Skripte zugreift. Die Struktur des Systems ist in Abbildung 23 gezeigt. Der Ablauf und die Informations- sowie Kontextstruktur erfüllen die in Unterabschnitt 3.2 designten Modelle. Insgesamt besteht das System aus mehreren Skripten, von denen *main.py* die Hauptanwendung darstellt. Dieses Skript bildet das übergeordnete Gerüst der Anwendung und organisiert die Ausführung der jeweiligen Anwendungsfunktionen.

Interface Der erste und für die Durchführung aller Systembestandteile wichtigste Teil ist die Verarbeitung der durch das Interface übergebenen Informationen. Dabei werden die externen übergebenen Parameter eingelesen und in ein geeignetes Format gebracht. Weiterführend ermöglichen einige globale Parameter eine dynamische Anpassung der Merkmalszusammensetzung oder das k der Top-k der STS. Dies unterstützt die Evaluierung und erleichtert zukünftige Anpassungen der Klassifikation. Das Ergebnis aus der Vorhersage des Zuordnungssystems wird über das Interface zurückgegeben.

Konfiguration Ein Bestandteil der übermittelten externen Parameter ist ein Konfigurationsstring. Dieser ist maßgeblich für die Ausführung der jeweiligen Codeabschnitte und in Abbildung 23 grau hinterlegt. Die Bezeichner auf den ausgehenden Pfeilen verkörpern den Inhalt der jeweiligen Zeichenkette. Ist einer dieser Strings in der übertragenen Konfiguration enthalten, wird der jeweilige Abschnitt ausgeführt. Dies sorgt für eine klare Trennung der verantwortlichen Code-Abschnitte und spart durch eine gezielte Ansprechung der Funktionen die benötigten Ressourcen. Die Bezeichner und ihre Funktionen, die für den normalen Betrieb verwendet werden

können, sind in Tabelle 5 zusammengefasst. Es gibt zusätzlich weitere Konfigurationen für eine detailliertere Evaluierung, diese werden hier jedoch nicht weiter betrachtet. Sie sind im Code selbst hinterlegt.

Tabelle 5: Übersicht über mögliche Konfigurationen im normalen Betrieb und deren Beschreibung

Konfig-Bezeichner	Beschreibung
edit_db	Vorverarbeitung des BMV und Erstellung bzw. Update der Vektordatenbank
load_FTS	Nur zur Vorbereitung der Trainingsdaten: Laden und Vorverarbeiten der FTS sowie Erstellung einer Excel-Tabelle zur Vorbereitung des Labelns
similarity	Nur zur Eignungs-Prüfung: Berechnung und Speicherung der STS Ergebnisse
train_svm	Vorbereitung der Trainingsdaten und Berechnung des Merkmalsvektors inkl. STS
train_svm_omly	Die SVM wird nur trainiert und ein kleiner Trainingsreport erstellt
predict	Laden des jeweiligen Modells und Vektordatenbank in Abhängigkeit des FS und Klassifikation des Testschrittes

Die jeweiligen Code-Abschnitte in *main.py* beziehen die meisten Funktionen aus weiteren Skripten. Diese sind ihren Funktionen nach unterteilt und in Abbildung 23 auf der rechten Seite im Anschluss an die zugehörige Konfiguration dargestellt. Auch hier sind nur solche Bestandteile dargestellt, die Teil des normalen Trainings- oder Klassifikationsprozesses sind. Solche Funktionen, die für die Evaluierung zuständig sind, werden nicht aufgeführt, da sie an dieser Stelle den Rahmen sprengen.

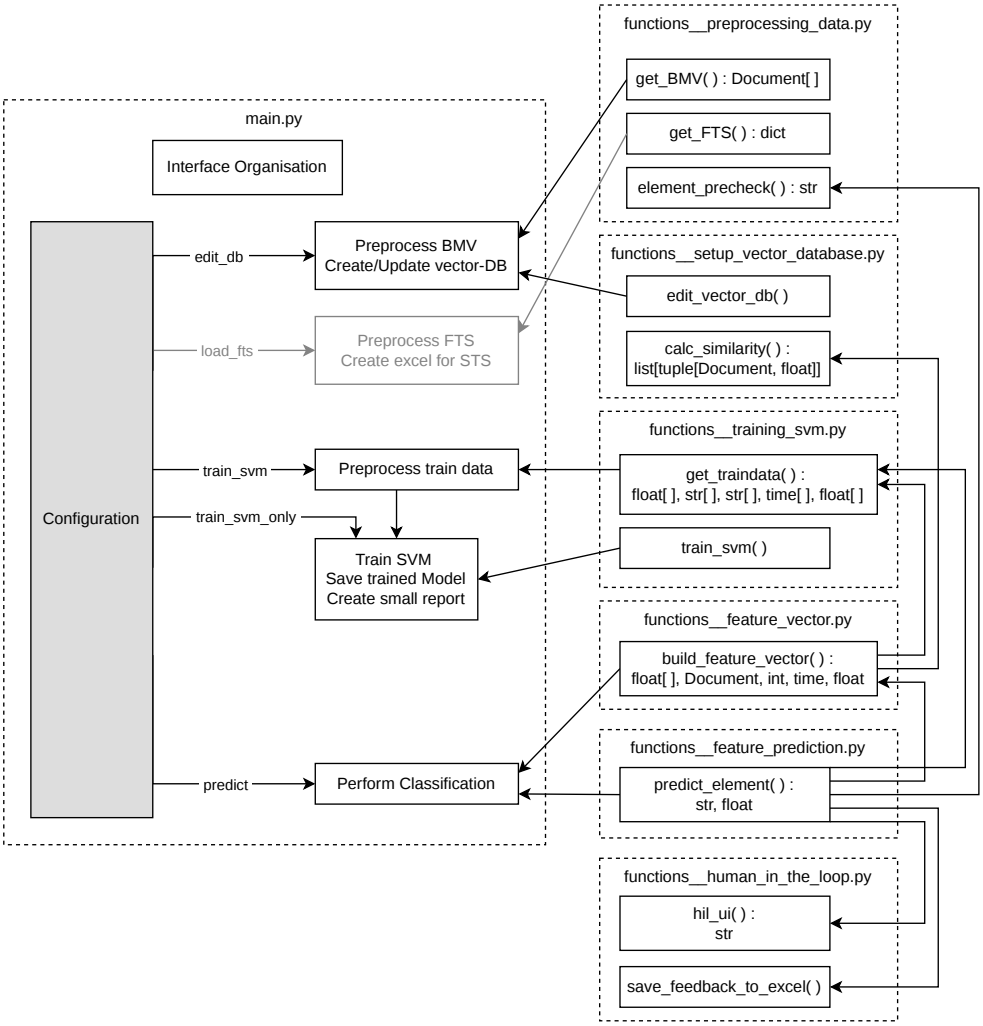

Abbildung 23: Übersicht über die Code-Struktur des Zuordnungssystems; Pfeilrichtung: wird genutzt von <...>

Externe Ressourcen Die externen Ressourcen wie das Transformer-Modell, die Vektordatenbank oder die trainierte SVM inklusive den Skalierer liegen als eigenständige Dateien im Projektverzeichnis ab. Da das Transformer-Modell zu groß für eine permanente Ablage im Versionierungssystem ist und sich eine Chroma-Vektordatenbank nicht auf andere Geräte portieren lässt, werden diese initial bei Nichtvorhandensein angelegt und lokal abgespeichert. Die Datenquellen und deren Ablage sind in Abbildung 24 dargestellt.

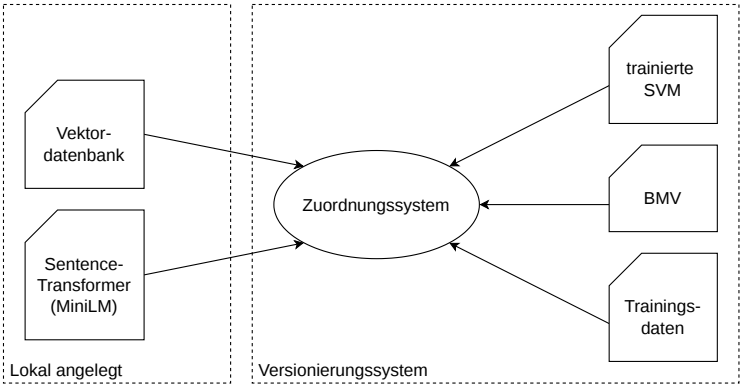

Abbildung 24: Ablage externer Ressourcen

Prüfinstanz Bevor das System mit der Vorhersage des Bedienelements beginnt, erfolgt eine Prüfung, ob ein Bezeichner des BMV nicht bereits im jeweiligen Testschritt-Text vorhanden ist. Sollte dies der Fall sein, wird der Bezeichner als korrektes Bedienelement akzeptiert. Da die FTS vor dem Einsatz in Validierungstests einem Review unterliegt, wird das enthaltene Element als korrekt angenommen. Für den Fall, dass kein Elemente vorhanden ist und die ML-Vorhersage durchgeführt wird, ermittelt das Programm einen Konfidenzwert des vorhergesagten Ergebnisses. Da aus der Vorhersage selbst keine relative Wahrscheinlichkeit für das Ergebnis ermittelt wird, erfolgt die Berechnung der Konfidenz über eine Softmax-Funktion. Sei dabei

$$s = (s_1, s_2, \ldots, s_K) \tag{46}$$

der Vektor der SVM-Scores der Klassen $K \in [1, 28]$, dann ist der Softmax

$$\mathrm{softmax}(s)_k = \frac{e^{s_k}}{\sum_{i=1}^{K} e^{s_i}}, \tag{47}$$

und somit

$$\mathrm{konfidenz} = \max(\mathrm{softmax}(s)_k), \tag{48}$$

wobei die Konfidenz den höchsten ermittelten Wert darstellt. Diese Metrik wurde gewählt, da sie auch bei einer Veränderung der Klassenanzahl vergleichsweise stabil bleibt. Ansonsten müsste der Grenzwert bei jeder Erweiterung des Modells angepasst werden. Auf Basis der Häufigkeitsverteilung der Konfidenzwerte (siehe Abbildung 25) durch eine Vorhersage mithilfe der Trainingsdaten wird der Wert 0,6 als Grenzwert ausgewählt.

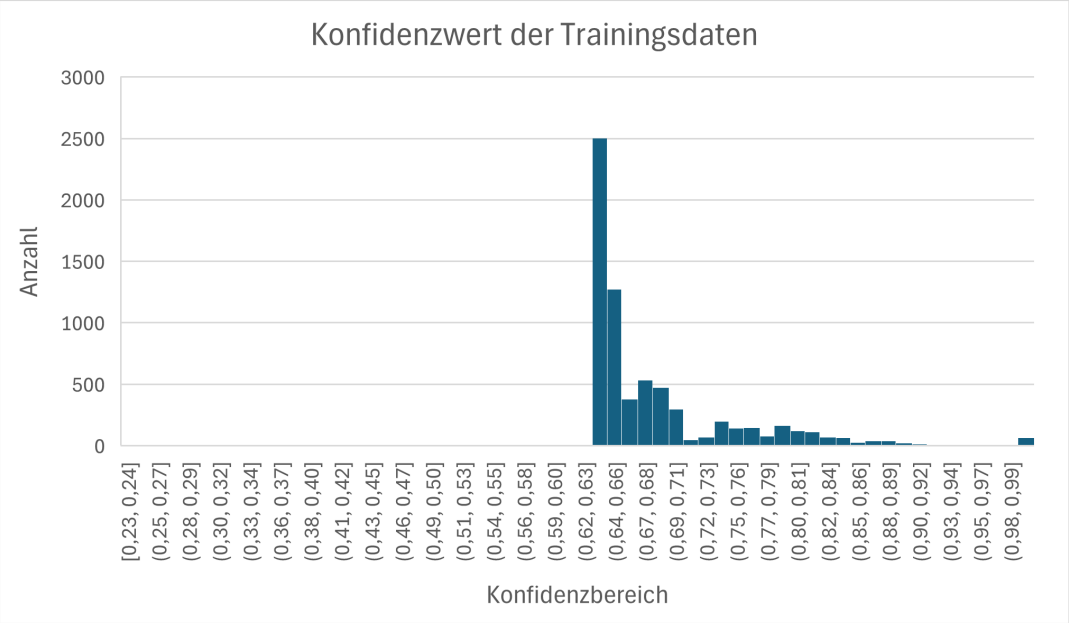

Abbildung 25: Häufigkeitsverteilung der Konfidenzwerte

4.2 Integration in bestehende Toolchain

Um das im Zuge dieser Arbeit erstellte Pythonprojekt im Kontextsystem nutzbar zu machen, muss es in das bestehende, prototypische Pythonprojekt integriert werden. Im Folgenden wird die dafür genutzte Methode vorgestellt sowie die Art und Weise, wie das fremde Projekt das neue Projekt einbindet und die Schnittstelle anspricht.

Virtuelle Umgebung Zum Zweck der Portierbarkeit wird eine virtuelle Umgebung [18] (engl. virtual environment (venv)) genutzt. Diese Umgebung stellt alles bereit, was das Projekt für eine reibungslose Ausführung benötigt. Inbegriffen ist dabei eine Kopie des verwendeten Interpreters sowie alle benötigten Pakete, die nicht in der Standardbibliothek enthalten sind. Damit wird sichergestellt, dass das Projekt immer rechnerunabhängig mit den vorhergesehenen Versionen arbeitet. Um das in dieser Arbeit entwickelte Projekt auf anderen Rechnern lauffähig zu machen, wird lediglich der Quellcode sowie eine Liste der Abhängigkeiten (Dateiname *requirements.txt*) übertragen. Mithilfe dieser Abhängigkeiten wird das venv auf dem Zielrechner neu erzeugt. Die benötigten Pakete werden dadurch automatisch in der wahlweise neusten oder benötigten Version installiert und stehen dem Projekt somit unmittelbar zur Verfügung.

Der Vorteil eines venv ist dabei, dass nur genau solche Pakete installiert werden, die das Projekt benötigt und es kommt nicht zu Konflikten mit bestehenden Paketen. Dies ist vor allem auf gemeinsam genutzten Rechnern relevant. Zudem

lassen sich so verschiedene Versionen der selben Bibliothek nutzen, ohne dass sie sich gegenseitig stören. Eine Voraussetzung für ein venv ist allerdings ein unlimitierter Internetzugang, um die Pakete zu installieren. Ist dies nicht der Fall, lassen sich die Pakete auch als Datei herunterladen und portieren, dies erhöht jedoch den Speicherbedarf bei der Übertragung auf andere Rechner.

Einbinden in fremde Projekte Bevor das Projekt von anderen Pythonprojekten genutzt werden kann, muss das Projekt inklusive einer stabilen Schnittstelle eingebunden werden. Dafür wird eine parallele Implementierung zu dem aktuell genutzten Fuzzy-Stringvergleich integriert, die je nach Bedarf an- und abgeschaltet werden kann. So bleibt die ursprüngliche Implementierung erhalten. An dieser Stelle werden zunächst zwei Parameter angelegt. Diese enthalten den Pfad zu der virtuellen Umgebung und der Pfad zu der aufzurufenden Codedatei des einzubindenden Projektes. Die anderen drei Parameter, die angelegt werden müssen, stellen die Interfaceparameter dar. Diese bestehen aus der FT-Nummer, dem Testschritt sowie der Konfiguration, mit welcher das Programm laufen soll. Der FT sowie der Testschritt werden dem jeweiligen Signal in dem betroffenen Codeabschnitt entnommen. Die Konfiguration wird manuell auf *predict* gesetzt. Diese fünf Parameter werden zusammen einem Subprozess übergeben, welcher das Zuordnungsprogramm ausführt. Das Ergebnis der Klassifikation wird über eine *stdout*-Ausgabe zurückgegeben, sodass das prototypische Programm damit weiterarbeiten kann.

4.3 Zusammenfassung

Dieses Kapitel beschrieb die praktische Umsetzung der zuvor modellierten Architektur. Die Implementierung wurde gekennzeichnet durch klar abgegrenzte Funktionsbereiche wie die Datenvorbereitung, das Vektordatenbank-Management, das SVM-Training und die Bedienelementvorhersage. Eine Konfigurationssteuerung ermöglicht dabei das gezielte Ansprechen bestimmter Funktionen, was sowohl die Integration als auch eine spätere Wartung effizient unterstützt. Die relevanten externen Ressourcen wie das Transformer-Modell, die Vektordatenbank und die trainierte SVM werden lokal verwaltet, um die Unabhängigkeit von externen Servern und die Einhaltung betrieblicher Sicherheitsanforderungen sicherzustellen. Eine integrierte Prüfung identifiziert zudem im Voraus bereits vorhandene E3-Namen im Testschritt und reduziert so die Anzahl der notwendigen ML-Vorhersagen. Des Weiteren erläuterte das Kapitel die Einbettung des erstellten Systems in die bestehende Toolchain. Dafür wird eine virtuelle Umgebung sowie eine Subprozessanbindung genutzt, die es erlaubt, das erstellte System modular zu integrieren.

5 Evaluierung

Nach der Modellierung und Implementierung des Systems erfolgt dessen Evaluation. Das nachfolgende Kapitel widmet sich der systematischen Bewertung der erzielten Ergebnisse, um die Leistungsfähigkeit, Robustheit und Aussagekraft des Modells zu überprüfen. Dabei befassen sich die Unterkapitel jeweils mit etablierten Methoden und Verfahren, die eine objektive Einschätzung und Analyse der Ergebnisse ermöglichen. Zu Beginn erfolgt eine Vorstellung der verwendeten Verfahren zur Erhebung analytischer Größen. Im Anschluss werden die verschiedenen Bestandteile des Zuordnungssystem untersucht, allen voran die Zusammensetzung des Merkmalsvektors. Diese ist in Unterabschnitt 3.1.3 bereits vorgestellt worden, jedoch ist der endgültige Aufbau von den Ergebnissen der Evaluierung abhängig. Ebenso wird die Nutzung des linearen Kernels bestätigt sowie der Hyperparameter C experimentell ermittelt. Weiterführend erfolgt die Betrachtung der Performance des Systems. Da es sich um eine Auslegung für geringe Hardwareressourcen handelt, ist der Speicherbedarf sowie die CPU-Auslastung ein Faktor, der vor allem bei Anwendungen mit ML-Komponenten ins Gewicht fällt. Die Ausführungsgeschwindigkeit, allen voran die Inferenzzeit, ist eine weitere relevante Größe bei der Verwendung in einem interaktiven System, wie es hier vorliegt. Im weiteren Verlauf erfolgt die Untersuchung der Modellqualität. Diese ist vor allem im Kontext der FuSi relevant und erfordert entsprechende Absicherung für falsch-positive Vorhersageergebnisse. Zuletzt wird das Modell auf seine Robustheit hin überprüft. Da das System auch projektübergreifend angewendet werden soll, sind Schwankungen in den Testschritten nicht auszuschließen. Da die Testanweisungen per Hand verfasst werden, sind auch hier Textfehler zu erwarten, was durch ein robustes Modell abgefedert wird. Generell erfolgt die Evaluation auf Basis eines geteilten Datensatzes [43], welcher in Trainings- und Testdaten unterteilt ist. Die Trainingsdaten werden zum Trainieren des Modells verwendet, die Testdaten zur Evaluierung der Ergebnisse.

5.1 Verwendete Methoden

In den folgenden Unterkapiteln werden solche Methoden vorgestellt, die zur Erhebung der Evaluierungsgrößen verwendet werden. Solche Größen, die einer ausführlicheren Erklärung bedürfen, werden zusätzlich vorgestellt. Die erklärten Methoden und Größen beinhalten die Cross-Validation für eine aussagekräftige Erhebung der Genauigkeit eines Modells. In diesem Zusammenhang werden die Größen *Precision*, *Recall* und *F1* vorgestellt sowie deren Abgrenzung zueinander erläutert. Zum Schluss wird die Confusion-Matrix erklärt sowie das Zustandekommen einer Modell-Lernkurve und aus welchen Verläufen sich welche Aussagen über das Modell ableiten lassen.

5.1.1 Cross-Validation

Die Cross-Validation [43] ist eine Methode zur Messung der Genauigkeit des Modells, indem das Training mehrmals und mit jeweils einer anderen Zusammensetzung der Trainings- und Testdaten durchgeführt wird. Die Trainings- und Testdaten werden dabei kalkuliert ausgewählt. Wie in Abbildung 26 abgebildet ist, wird der Datensatz *D* in *n* gleich große Teile eingeteilt. Jeweils ein Datenpaket (entspricht (*n-1*)/*n* von D) wird als Testdaten verwendet, der Rest als Trainingsdaten. Die Fehlerberechnung wird somit *n* mal durchgeführt, wobei in jedem Durchlauf ein anderes Datenpaket als Testdaten fungiert. Für die vorliegende Anwendung der Multiklassen-Klassifikation erweisen sich diese Metriken als geeignete Evaluierungsgrößen, da sie eine direkte Aussage über die Gesamtklassifikationsleistung des Modells ermöglichen.

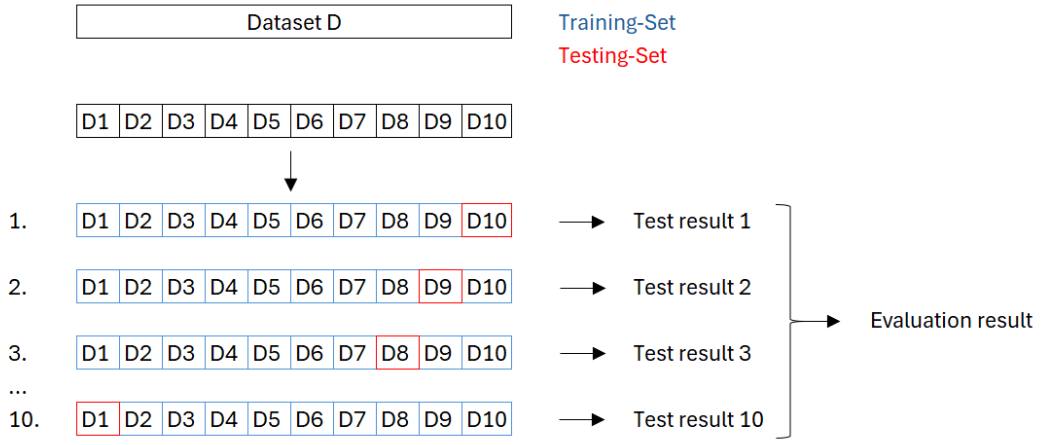

Abbildung 26: Darstellung der Cross-Validation Methode mit n=10. Rote Kästchen: Testdaten, blaue Kästchen: Trainingsdaten

5.1.2 Genauigkeit

Bezüglich der Effizienz-Analyse gibt es verschiedene Möglichkeiten, ein ML-Modell auf seine Qualität hin zu untersuchen. Eine bewährte Methode besteht in der Analyse der Fehlerrate und der Genauigkeit [42] des Modells. Sie sind ein Maß dafür, wie viele Vorhersagen des Modells in Bezug auf die Menge aller Vorhersagen falsch beziehungsweise korrekt getroffen werden. Es ist dabei bedeutend, ob die Grundlage der Fehlerberechnung in dem Trainingsdatensatz oder dem Testdatensatz liegt. Für eine Evaluierung werden gesonderte Testdaten verwendet. Da das Modell hier auf einen unbekannten Dateninput reagieren muss, ist die Reaktion realitätsbezogener und kann von dem Fehler der Trainingsdaten abweichen. Weiterführend ist die Fehlerrate ebenfalls ein mögliches Indiz für die Qualität der Trainingsdaten. Ist beispielsweise der Fehler der Trainingsdaten sehr

gering, der Fehler der Testdaten jedoch sehr hoch, ist von einem Oversampling [42] auszugehen. Die Trainingsdaten sind in diesem Fall nicht ausgeglichen genug, sodass das Modell auf die bekannten Muster des Trainings reagiert, jedoch auf unbekannte Daten nicht die nötige Flexibilität besitzt.

Eine detailliertere Betrachtung der Genauigkeit lässt sich über die Metriken *F1*, *Precision* und *Recall* [15] durchführen. Sie geben Auskünfte über die Verteilung richtig-positiver (engl. true-positive TP), falsch-positiver (engl. false-positive FP) und falsch-negativer (engl. false-negative FN) Vorhersagen. Der F1-Score entstammt dabei dem harmonischen Mittel aus Precision und Recall:

$$\text{Precision} = \frac{TP}{TP + FP}, \qquad \text{Recall} = \frac{TP}{TP + FN} \tag{49}$$

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{50}$$

Allgemein bedeutet eine höhere Precision in der Vorhersage, dass es mehr falsch-negative als falsch-positive Ergebnisse gibt. Das Modell ist konservativer in der Entscheidung für eine Klasse, was darauf hinweist, dass die Entscheidungsgrenze bei dieser Klasse unklarer ist als bei anderen. Der umgekehrte Fall, dass der Recall größer ist als die Precision, spricht für eine großzügigere Entscheidungsgrenze, in welcher mehr falsch-positive Ergebnisse toleriert werden.

5.1.3 Confusion-Matrix

Eine Confusion-Matrix [15] ist eine Tabelle, die eine grafische Repräsentation der Vorhersagequalität eines Modells darstellt. Dabei werden die tatsächlichen Klassen mit den vom Modell vorhergesagten Klassen verglichen. Es liefert somit detaillierte Informationen darüber, welche Fehler das Modell macht und welche Klassen eher zur Verwechslung oder falscher Zuordnung tendieren. Auf Basis der Ergebnisse einer Confusion-Matrix lassen sich zudem Metriken wie Precision, Recall und F1-Score berechnen, wie es bereits in Unterabschnitt 5.1.2 ausgeführt wurde.

Das Prinzip einer Confusion-Matrix ist in Abbildung 27 durch eine binäre Klassifikation in Bild a) gezeigt. Die vorhergesagten Klassen werden den tatsächlichen Klassen gegenübergestellt, was in einer Verteilung der folgenden Ergebniszustände resultiert:

- TP: richtig-positiv (engl. true-positive)
- TN: richtig-negativ (engl. true-negative)
- FP: falsch-positiv (engl. false-positive)
- FN: falsch-negativ (engl. false-negative)

Die Anzahl der entsprechenden Ergebnisse wird in den jeweiligen Feldern notiert, wie es in Abbildung 27 in Bild b) gezeigt ist. Wird das Ergebnis zusätzlich seines

Wertes entsprechend eingefärbt, lassen sich auf einen Blick die Genauigkeit und Fehlertendenz einer Klasse ablesen. Hohe Werte im FP-Bereich verschlechtern die Precision der Klasse und hohe Werte im FN-Bereich verschlechtern den Recall. Für eine Mehrklassen-Klassifikation wie im Bild c) der Abbildung 27 gilt dasselbe Prinzip, nur dass es mehr Klassen gibt, die einander gegenübergestellt werden.

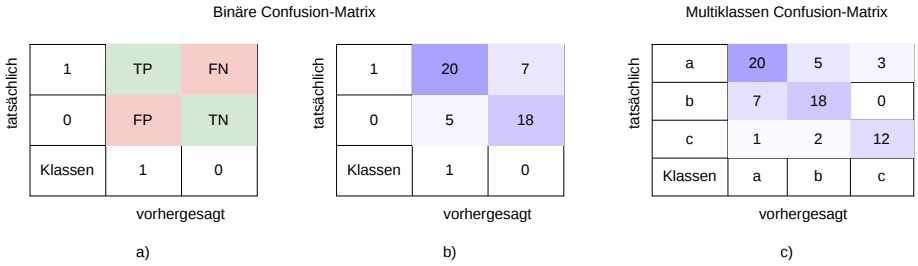

Abbildung 27: Methodische Darstellung einer Confusion-Matrix für binäre und multiklassen Klassifikationen

5.1.4 Lernkurve

Die Lernkurve [28] eines Modells zeigt, wie sich die Leistung eines ML-Modells in Abhängigkeit der verfügbaren Trainingsdaten entwickelt. Aus der Kurve lässt sich ablesen, ob dem Modell für eine gute Generalisierbarkeit ausreichend Trainingsdaten zur Verfügung stehen und ob es über- oder unterangepasst (engl. over-/underfitting) ist.

Eine Lernkurve besteht aus zwei Linien, eine für die Trainingsgenauigkeit und eine für die Validierungsgenauigkeit. Für die Fehlerrate sind die Kurven entsprechend invertiert. Die Trainingsgenauigkeit bezieht sich darauf, wie gut das Modell mit bekannten Daten performt. Bei der Validierungsgenauigkeit bezieht es sich analog auf unbekannte Daten. Beide Kurven sind in Abhängigkeit der bereits verarbeiteten Trainingsdatenmenge aufgetragen. Der Kurvenverlauf ergibt dabei folgende Muster und daraus folgende Eigenschaften:

- **Unteranpassung**: Trainingsgenauigkeit niedrig, Validierungsgenauigkeit niedrig, Modell ist zu einfach
- **Überanpassung**: Trainingsgenauigkeit hoch, Validierungsgenauigkeit niedrig, Modell ist zu komplex
- **Optimal**: Beide Kurven konvergieren zu hohen Werten, kleiner Abstand der Kurven

5.2 Merkmalsvektor

Im Zuge der Evaluierung werden verschiedene Zusammensetzungen des Merkmalsvektors getestet. Eine Übersicht mit den entsprechenden Bezeichnungen der Varianten ist in folgender Tabelle 6 zusammengefasst. Dabei muss beachtet werden, dass jede Zusammensetzung für jeweils die Top-k Ergebnisse der STS mit $k \in \{3, 5, 7, 10, alle\}$ stattfindet, um auch hier das optimale Maß zu ermitteln. Die getesteten Merkmale bestehen dabei aus den folgenden Metriken:

- Emb: Textembedding des Testschritts
- WEmb: Gewichtetes Embedding
- STS: k STS-Scores
- Cos: k Kosinus-Distanzen
- MS: Mittelwert der k STS-Scores
- VS: Varianz der k STS-Scores
- MC: Mittelwert der k Kosinus-Distanzen
- VC: Varianz der k Kosinus-Distanzen
- MinS: minimale STS-Score
- MaxS: maximale STS-Score
- RS: Wertebereich der k STS-Scores
- RelS: Verhältnisse der k STS-Scores

Tabelle 6: Übersicht der Zusammensetzungen des Merkmalsvektors für die Evaluierung und deren Kürzel (richtet sich nach der Anzahl der Merkmale)

Kürzel	Merkmale								
	Emb.	STS	Cos	MS	MC	VS	VC		
2feat		x	x						
6feat		x	x	x	x	x	x		
	Emb.	WEmb	STS	MS	VS	MinS	MaxS	RS	RelS
8feat		x	x	x	x	x	x	x	x
9feat	(384)x	x	x	x	x	x	x	x	x

5.2.1 Klassenzusammensetzung

In Hinblick auf den Aufwand des Labelns und der Trainingsdauer für die Evaluierung, werden nicht alle möglichen Klassen des BMV zur Evaluierung des Modells herangezogen. So können in zeitlicher Hinsicht mehr Daten pro Klasse erhoben und analysiert werden. Die verwendeten Bedienelemente beschränken sich auf solche, die im Kontext der FuSi-Tests den größten Stellenwert einnehmen und am häufigsten verwendet werden. Sie sind in Tabelle 7 zusammengefasst. Der Reihenfolge und der Gruppierung werden dabei keine besondere Bedeutung beigemessen. Das Kürzel richtet sich nach der Anzahl der enthaltenen Klassen in der jeweiligen Zusammensetzung (durch die zwei FTs das doppelte der Anzahl der Bedienelemente)

Da viele Funktionen eine Aufrüstung (fahrbereit machen) der Lokomotive erfordern, werden die hier verwendeten Bedienelemente und deren Konterpart ermittelt. Der Konterpart erzeugt zwar weniger Vielfalt der verschiedenen Klassen, jedoch sorgt der feine textuelle Unterschied zwischen beispielsweise „Occupy cab via pushbutton “ und „Deoccupy cab via pushbutton “ für einen sehr großen semantischen Bedeutungsunterschied, welche die SVM trennen können muss.

Tabelle 7: Auflistung der Klassenzusammensetzung inklusive Bezeichner

Kürzel	Bedienelement	Bezeichner (FT1/FT2)
4class	Besetzen des Führerstandes	-GC.S01/S02
	Entbesetzen des Führerstandes	-GC.S03/S04
8class	Lok-Batterie anschalten	-HC.S01/S02
	Lok-Batterie ausschalten	-HC.S03/S04
12class	Lösen der Parkbremse	-GC.S21/S22
	Bedienung des Fahr-Brems-Hebels (TBC)	-TD.A23/A24-A01
16class	Heben und Senken des Pantographen	-TD.A23/A24-S02
	Lösen/Aktivieren der automatischen Bremse	-GC.S29/S30
20class	Auswahl der Fahrtrichtung "vorwärts"	-TD.A21/A22-S12
	Auswahl der Fahrtrichtung "neutral"	-TD.A21/A22-S13
24class	Auswahl der Fahrtrichtung "rückwärts"	-TD.A21/A22-S14
	Bedienung des Notbrems-Knopf	-GC.S07/S08
28class	Bedienung des Sicherheits-Fahrschalters (SiFa)	-TD.A01/A02-S01
	Bedienung des Hauptschalter-Hebels	-TD.A23/A24-S07

5.2.2 Merkmalszusammensetzung

Der nachfolgende Vergleich in Abbildung 28 wird mit einer Zusammensetzung nach Abbildung 28 durchgeführt. Dies sorgt für eine moderate Trainingszeit, aber enthält bereits genug Daten, um die nötige Aussagekraft zu besitzen.

Tabelle 8: Trainingskonfiguration der grundlegenden Merkmalsanalyse

Kernel	linear, C=1
Trainingsdaten pro Klassen	80
Evaluierungsdaten pro Klasse	20
Top-k	k=3

Die Auswertung findet mittels einer Confusion-Matrix (siehe Unterabschnitt 5.1.3) statt. Diese trägt die erwarteten Klassen gegen die tatsächlich vorhergesagten Klassen auf. Bei einer perfekten Zuordnung bildet sich eine Diagonale in der Matrix. Die farbliche Ausprägung der Einträge gibt die Menge der Zuordnungen auf dem jeweiligen Feld zurück.

In der Abbildung 28 ist ersichtlich, dass sich das Modell, welches mit dem Bezeichner *9feat* trainiert wurde, deutlich von den anderen Varianten abhebt. Der hier verwendete Vektor ist der einzige, der das Text-Embedding der Eingabe in den Merkmalsvektor einbezieht. Dies stellt somit ein signifikant aussagekräftiges Merkmal dar. Im Vergleich unter den drei anderen Zusammensetzungen besitzt *8feat* eine besser ausgeprägte Diagonale als *2feat* und *4feat*. Aufgrund der Betrachtung mehrerer Statistiken des STS-Scores in *8feat*, unter anderem der relativen Verhältnisse, ist dieses Ergebnis zu erwarten gewesen. Die inhaltliche Zusammensetzung des Merkmalsvektors wird somit auf *9feat* festgelegt.

Abbildung 28: Vergleich verschiedener Merkmals-Zusammenstellungen in Form von Confusion-Matrizen der Trainings-Ergebnisse; y-Achse: erwartete Klassen, x-Achse: vorhergesagte Klassen

Bedeutungsanalyse Des Weiteren wird ermittelt, wie sich das Modell bei dem Entfernen einzelner Merkmale aus dem Merkmalsvektor verhält. So lässt sich der Einfluss einzelner Merkmale auf die Modell-Qualität untersuchen und gegebenenfalls anpassen. Aufgrund der fortgeschrittenen Evaluierung auch im Bereich der STS, wird mit angepassten Parametern gearbeitet, wie sie in Tabelle 9 zusammengefasst sind.

Tabelle 9: Trainingskonfiguration der detaillierten Merkmalsanalyse

Kernel	linear, C=1
Trainingsdaten pro Klassen	120
Evaluierungsdaten pro Klasse	30
Top-k	k=5

Anhand der Abbildung 29 lässt sich schnell erkennen, dass das Embedding des Testschritt-Textes den Merkmalsvektor in seiner Klassifikations-Fähigkeit dominiert. Das Merkmal ist notwendig für eine hohe Genauigkeit. Ohne die Einbeziehung des Embeddings bricht die Klassifikation ein. Die anderen Merkmale verändern die Genauigkeit des Modells bei Abwesenheit nicht mehr als 0,03 % (vergleiche mit dem grünen Kontrollbalken in Abbildung 29). Dies ist ein Hinweis darauf, dass der Vektor robust genug ist, um einzelne starke Schwankungen auszugleichen. Andererseits ist es jedoch auch ein Hinweis auf Redundanz der einzelnen STS-Merkmale, da deren einzelner direkter Einfluss auf die Modellleistung bereits durch andere Merkmale übernommen wird. Der Zusatznutzen der einzelnen STS-Metriken für die Genauigkeit der Klassifikation scheint somit marginal zu sein. Bei Betrachtung des Cross-Validation Scores für den Merkmalsvektor, welcher nur aus dem Text-Embedding besteht (lila Balken in Abbildung 29), ist der Score 0,06 % besser als der des Baseline-Vektors. Dies kann ein Hinweis darauf sein, dass die Ergebnisse der STS keinen unmittelbaren Zusatznutzen bei der Klassifikation bringen. Dabei muss jedoch auch beachtet werden, dass die Metriken der STS nur 3,3 % des gesamten Merkmalumfangs ausmachen und ein Unterschied von 0,06 % somit sehr gering bis vernachlässigbar ist. Um eine bessere Aussage in diesem Zusammenhang treffen zu können, werden im Folgenden auch die Werte der Performance und Ressourcennutzung betrachtet.

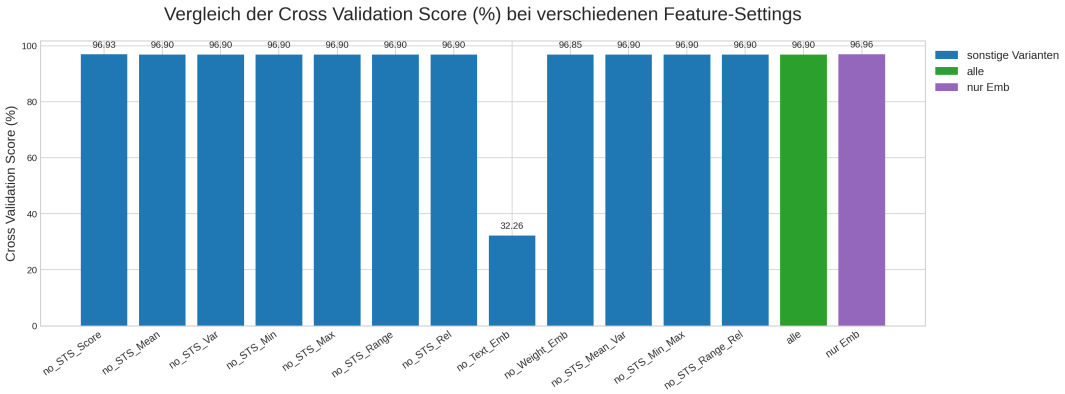

Abbildung 29: Auftragung der Cross-Validation Scores für den Merkmalsvektor mit dem Auslassen bestimmter Einträge; Baseline: grün, nur Embedding: lila

Trainingszeit Bei der Betrachtung der Trainingszeit und -ressourcen (siehe Abbildung 30 und Abbildung 31) sticht auch hier das Text-Embedding maßgebend heraus. Neben dem Text-Embedding sorgt das Weglassen einzelner Merkmale für keinen signifikante RAM-Einsparungen (maximal 0,06 MB = 60 kB). Es fällt jedoch erneut auf, dass der Vektor ohne die STS-Metriken die größte Abweichung nach unten mit fast 0,1 MB besitzt. Das entspricht ca. 1,5 % weniger RAM-Bedarf während des Trainings, was auch hier als vernachlässigbar betrachtet wird. Auch die Trainingszeit des Merkmalsvektors in Abbildung 31, welcher ausschließlich das Text-Embedding enthält, ist am geringsten jedoch auch nur mit ca. 3 %. Dies ist mehr als bei dem RAM-Vergleich, macht jedoch im Hinblick auf die Trainingszeit ebenfalls einen vernachlässigbaren Unterschied.

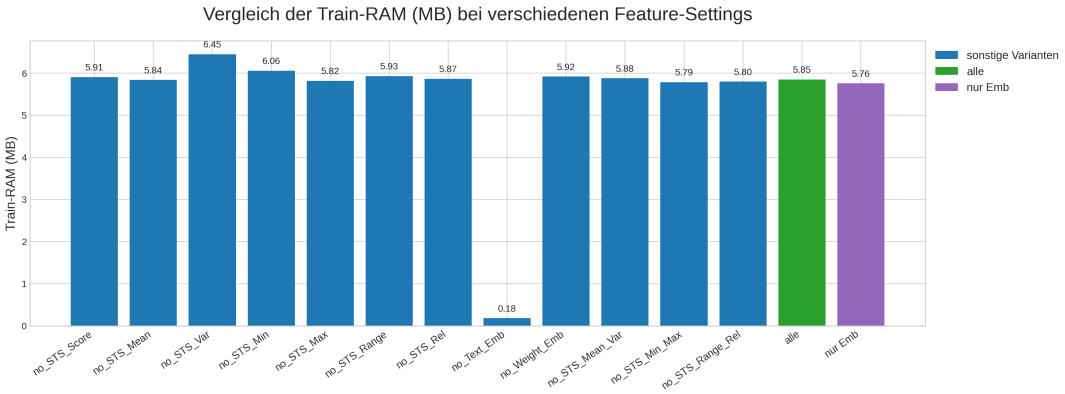

Abbildung 30: Auftragung des Trainings-RAM für den Merkmalsvektor mit dem Auslassen bestimmter Einträge

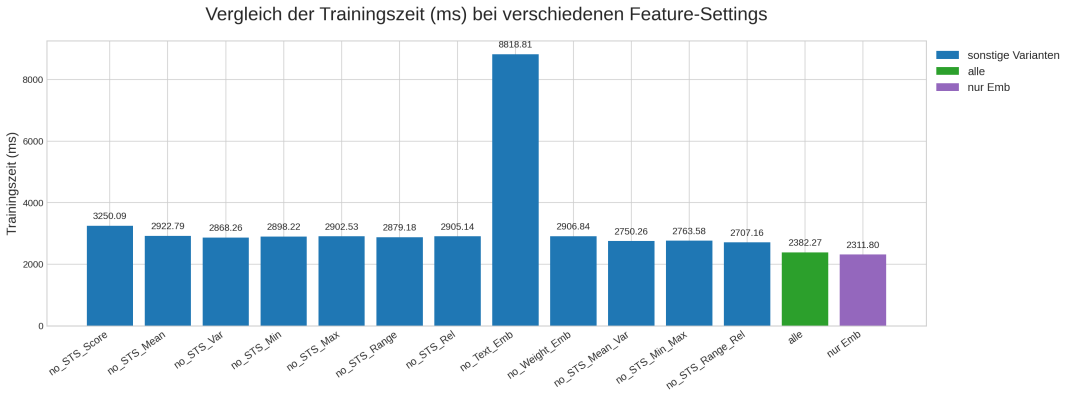

Abbildung 31: Auftragung der Trainingszeit für den Merkmalsvektor mit dem Auslassen bestimmter Einträge

Inferenzzeit Die Betrachtung der Inferenzzeit in Abbildung 32 liefert größere Differenzen in den Werten als zuvor. Die jeweiligen Scores der einzelnen Merkmale sind nicht so einheitlich, wie zuvor bei der Analyse des Trainings. Die meisten Werte liegen über den 287.42 ms der grünen Baseline-Messung, was für eine langsamere Inferenz durch weniger Informationen für die Klassifikation spricht. Einzig die Messungen mit weggelassenem STS-Verhältnis (no_STS_Rel) und dem gewichtetem Embedding (no_Weight_Emb) liegen 3-5 ms unter diesem Wert, was bei dieser Größenordnung auch systematischen Schwankungen zugrunde liegen kann. Die Inferenzzeit des Embedding-only-Vektors liegt bei 308.81 ms, was ca. 20 ms höher ist als der Wert des Baseline-Vektors. Im Hinblick auf die Relevanz der Inferenzzeit in dieser Arbeit, ist dies ein Indiz dafür, den ursprünglichen Vektor mit allen STS-Informationen zu bevorzugen.

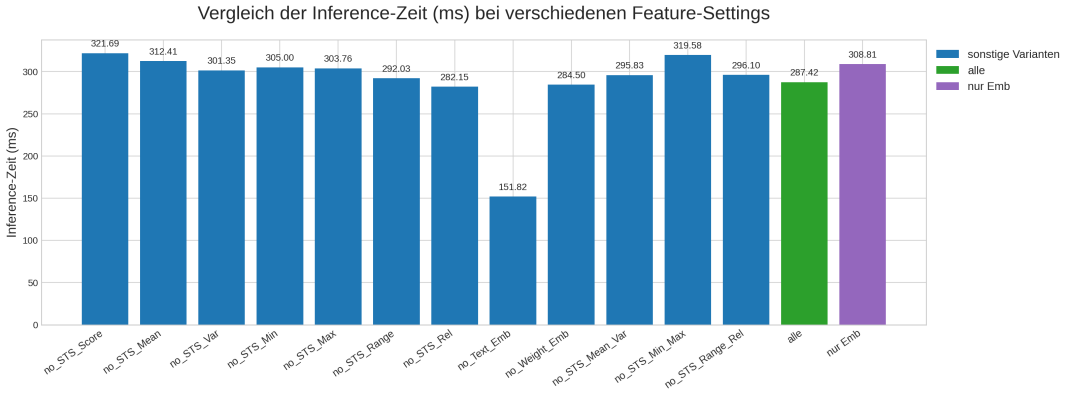

Abbildung 32: Auftragung der Inferenzzeit für den Merkmalsvektor mit dem Auslassen bestimmter Einträge

Klassenverlauf Im direkten Vergleich des ursprünglichen *9feat*-Vektors und des Embedding-only-Vektors in Abbildung 33 zeigt der Verlauf bei steigender Klassenzahl einen allgemein höheren Trend der Cross-Validation des Merkmalsvektors, welcher neben dem Text-Embedding auch die STS-Metriken beinhaltet. Lokal liegt der Cross-Validation-Score des Vektors ohne STS bei 28 Klassen höher, dies erscheint in dieser Auftragung jedoch nur eine punktuelle Situation darzustellen. Im Gesamtverlauf besitzt der Vektor mit allen Metriken den höheren Score.

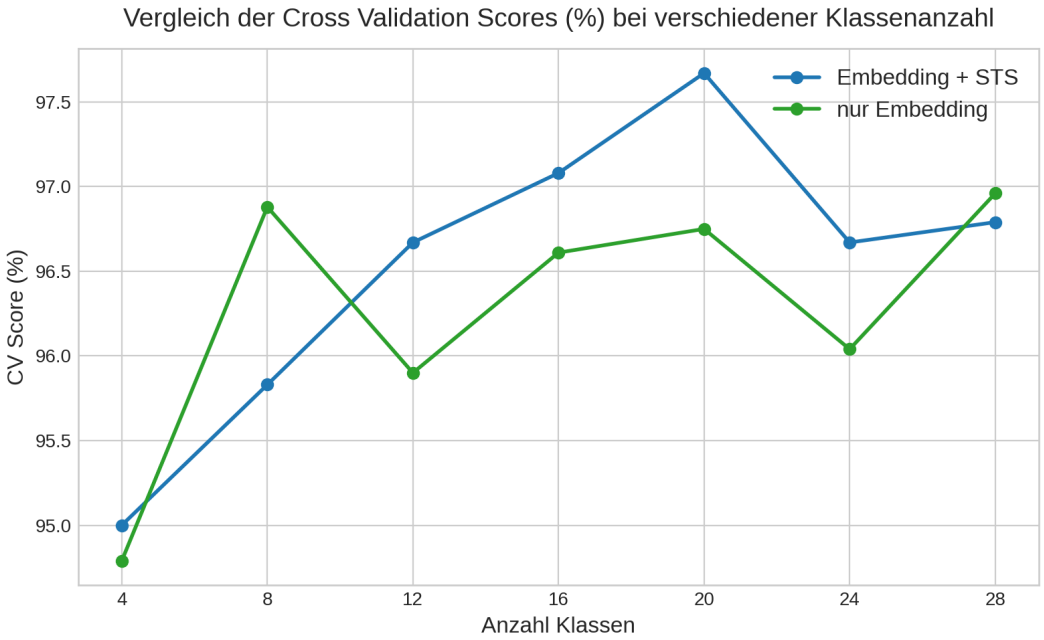

Abbildung 33: Vergleich der Cross-Validation-Scores über verschiedene Klassenanzahlen für den Merkmalsvektor mit und ohne die STS-Metriken

Zusammenfassend lässt sich beobachten, dass das Embedding des Eingabe-Textes den primären Informationsgehalt für eine Klassifikation besitzt. Die Zusatzmerkmale der STS Berechnung sind im Verhältnis zur Dimension des Embeddings anteilig sehr wenig und korrelieren zusätzlich mit den Informationen, die im Text-Embedding selbst enthalten sind. Dies wird gestützt durch den geringen Einfluss der einzelnen STS-Informationen auf den Cross-Validation-Score des Modells und deutet auf redundante Informationen hin. Dies kann vor allem bei wenig Trainingsdaten dazu beitragen, den Merkmalsvektor robuster zu machen, da die selbe Struktur präsenter im Merkmalsvektor auftritt. Andererseits fällt dies auch zulasten der Rechenressourcen, was bei der Analyse jedoch einen nur marginalen Einfluss gezeigt hat. Bezüglich der Inferenzzeit liegt die Schwankung mit 20 ms zum Baseline-Merkmalsvektor bei ca. 7,4 % und sollte nicht vernachlässigt werden. Der direkte Vergleich der Cross-Validation über verschiedene Klassenzahlen zeigt in seinem Trend eine stärkere Leistung des Merkmalsvektors, welcher die STS-Informationen beinhaltet. Diese Beobachtung zusammen mit der Zunahme der Inferenzzeit des Embedding-only-Vektors sprechen für eine weitere Verwendung des *9feat*-Vektors. Dennoch zeigt das Ergebnis dieser Analysen eindeutig, dass die Verbesserung des Merkmalsvektors weiterhin eine Herausforderung bleibt, die in der Zukunft weiter verfolgt werden sollte.

5.2.3 Anzahl STS-Ergebnisse

Die STS gibt eine sortierte Liste der Ähnlichkeitsscores (STS-Score) zwischen dem Testschritt und jedem Eintrag der Vektordatenbank zurück. Für den Merkmalsvektor ist folglich die Anzahl $k \in \mathbb{Z}$ der besten Ergebnisse relevant, die miteinbezogen werden. Zur Ermittlung des besten k der STS-Ergebnisse (Top-k), werden zunächst die Werte der Genauigkeit und parallel dazu der Cross-Validation betrachtet. Diese sind für die $k \in \{3, 5, 7, 14(alle)\}$ in Abbildung 35 aufgetragen. Es wird im Voraus davon ausgegangen, dass ein zu großes k zu viele irrelevante Informationen in den Merkmalsvektor einbringt und dadurch Rauschen erzeugt. Aus dem Grund wird ein größerer Fokus auf $k <= 7$ gelegt. Die Verteilung der Positionen der korrekten Labels in der Ergebnisliste der STS in Abbildung 34 verdeutlicht dies. Dort ist gut sichtbar, dass die meisten Treffer innerhalb der Top-5 liegen. Die Häufigkeit der Treffer im Ranking unterhalb der Top-5 sind vergleichsweise sehr gering. Aus diesem Grund wird die Top-5 in die Analyse als Kniepunkt mit einbezogen sowie die Top-3 als Mitte zwischen $k = 5$ und $k = 1$. Die Top-7 wird symmetrisch dazu im unteren Rankingbereich analysiert. Zusätzlich erfolgt die Betrachtung des gesamten Ranking-Ergebnisses der STS als Randfall.

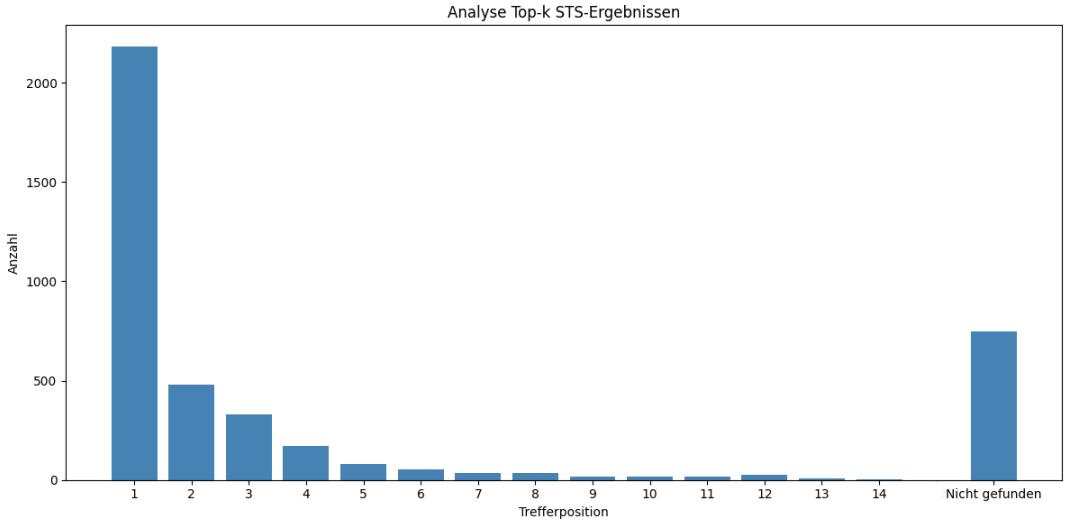

Abbildung 34: Verteilung der Ranking-Positionen des korrekten Labels im STS Ergebnis

Das Modell-Training erfolgt mit der Konfiguration, wie sie in Tabelle 10 abgebildet ist. 14 Bedienelemente ergeben unter Berücksichtigung der beiden Führerstände 28 Klassen und es wird die zuvor evaluierte Merkmalszusammensetzung *9feat* verwendet. Die 150 Trainingsdaten pro Klasse ergeben eine entsprechende Aufteilung bei einer klassischen Aufsplittung von 80 %-20 % für das Training und die Validierung.

Tabelle 10: Trainingskonfiguration der Top-k Analyse

Klassenzusammensetzung	28class
Merkmalszusammensetzung	9feat
Kernel	linear, C=10
Trainingsdaten pro Klassen	120
Evaluierungsdaten pro Klasse	30

In der Abbildung 35 ist abzulesen, dass der Cross-Validation-Score mit steigendem k ebenfalls ansteigt, jedoch gleichzeitig die Differenz zur Genauigkeit ebenfalls größer wird. Der Unterschied zwischen der Cross-Validation und der Genauigkeit liegt an sich in der Aufteilung des Trainings- und Validierungs-Datensatzes. Da die Verteilung bei jedem neuen Training anders ist, wird das Modell nicht immer mit denselben Daten trainiert. Dies sorgt dafür, dass die Genauigkeit einer solchen Zusammensetzung zwischen verschiedenen Trainings schwankt. Die Cross-Validation wird im Gegensatz auf verschiedene Zusammensetzungen trainiert und deckt somit mehr Varianten ab. Eine kleine Differenz zwischen beiden Größen bedeutet somit eine stabilere Leistung über verschiedene Datenaufteilungen hinweg. Da die Messung mehrfach vorgenommen und gemittelt wurde, ist bei der Genauigkeit eine zufällig ähnliche Verteilung des Datensatzes im Vergleich zur Cross-Validation ausgeschlossen. Bei einer großen Differenz der beiden Werte kann man von bestimmten Schwierigkeiten im Modelltraining ausgehen. In diesem Fall ist der Wert der Cross-Validation deutlich höher als die einfache Genauigkeit. Dies ist ein deutlicher Hinweis auf Überanpassung. Das Modell hat nur die Muster der Trainingsdaten gelernt und generalisiert schlecht auf neue Daten. So ist es nicht unwahrscheinlich, dass die Trainingsdaten noch zu gering für eine gute Generalisierung sind, es bedeutet aber auch, dass die Merkmale der Top-3 eine bessere Generalisierung zulassen und mit den vorhandenen Daten eine stabilere Vorhersage ermöglichen. Aus diesem Grund wird zunächst k = 3 als bester Wert für die STS verwendet.

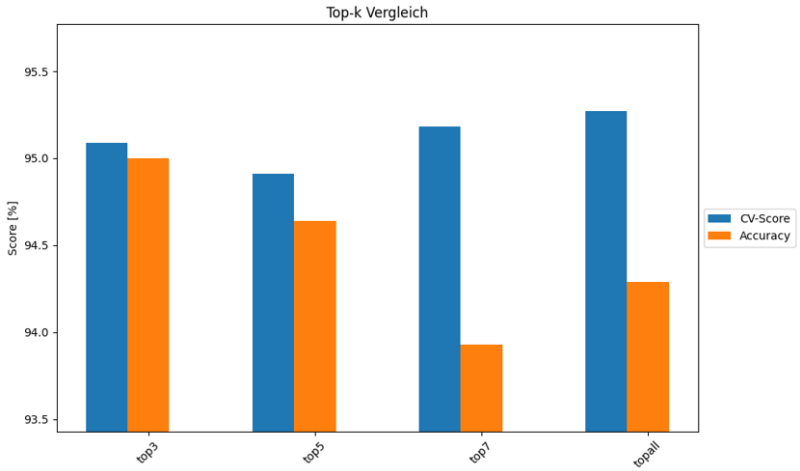

Abbildung 35: Auftragung der Cross-Validation (CV) und Genauigkeits-Scores für Top-k mit k=[3, 5, 7, alle]

Im Zuge der Erweiterung des Trainingsdatensatzes auf 150 Datenpaare pro Klasse lohnt sich die erweiterte Betrachtung der Auswirkung auf die verschiedenen Top-k. Neben der Untersuchung der Cross-Validierung als aussagekräftige Repräsentation der Modell-Genauigkeit wird des Weiteren sowohl die Trainings- und Inferenzzeit als auch die RAM-Nutzung betrachtet. Je nachdem, wie gut die Top-k Ergebnisse im Merkmalsvektor zur Klassifikation beitragen, beeinflussen sie dadurch auch die Komplexität der Berechnungen durch das Modell und somit auch die verwendete Zeit und Ressourcen. Für den erweiterten Vergleich wird das Modell zusätzlich mit einem k = 10 trainiert. Dieser Wert füllt den Raum zwischen k = 7 und k = alle, welcher als Zusatzinformation relevant zu betrachten ist. Zudem wird dieselbe Merkmals-Konfiguration wie zuvor verwendet, hier pro Klasse jedoch mit 150 Datenpaaren im Trainingsdatensatz, um die Generalisierungs-Fähigkeit des Modells zu verbessern.

Cross-Validierung Begonnen wird mit der Analyse der Cross-Validierungs Scores. Hier wird in Abbildung 36 ersichtlich, dass die Top-3, Top-5 und Top-7 die besten Trainingsergebnisse hervorbringen. Dazu muss jedoch beachtet werden, dass die Differenz zu den Scores der Top-10 und Top-alle nur 0,11 % beträgt. Diese Differenz ist sehr gering, dennoch werden die Top-k mit den besseren Scores in den weiteren Überlegungen bevorzugt betrachtet.

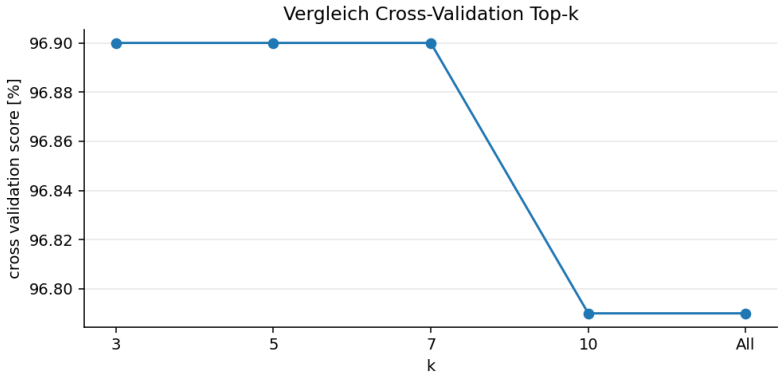

Abbildung 36: Auftragung der Cross-Validation Scores für Top-k mit k=[3, 5, 7, 10, alle]

Inferenzzeit Weiterführend erfolgt die Betrachtung der Inferenzzeit und RAM-Nutzung. Die RAM-Nutzung ist nicht primär von der Größe des Merkmalsvektors abhängig, sondern unter anderen von der Anzahl der Support-Vektoren, um die Klassen im Vektorraum gut zu trennen. Sind die Merkmale nicht gut gewählt, werden mehr Support-Vektoren benötigt, um die Klassen zu separieren. Sind die Merkmale gut separierend, werden weniger Support-Vektoren und es wird somit auch weniger RAM benötigt. In Abbildung 37 sind die RAM-Werte für Top-3, Top-5 und Top-10 nahezu konstant mit dem geringsten Wert von 0,04 MB bei Top-5. Top-7 ist mit 0,11 MB sichtbar höher und die RAM-Nutzung aller Ergebnisse der STS, erreicht mit 0,72 MB den höchsten Wert. Unter der Prämisse, dass eine geringe RAM-Nutzung mit einer guten separierenden Eigenschaft des Merkmalsvektors einher geht, ist Top-5 diesbezüglich am besten geeignet.

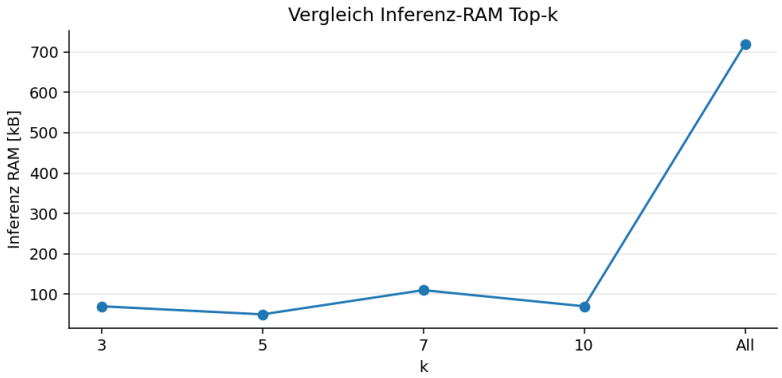

Abbildung 37: Auftragung der Inferenz-RAM für Top-k mit k=[3, 5, 7, 10, alle]

Die bisherige Beobachtung bezüglich der Eignung von $k = 5$, wird auch während der Betrachtung der Inferenzzeit in Abbildung 38 bestätigt. Zwar liegt die Inferenzzeit der Top-alle bei durchschnittlich 277 ms, der Wert der Top-5 liegt jedoch mit ca. 288 ms immer noch unter dem der anderen. Unter der Berücksichtigung, dass die Werte der verschiedenen Messungen immer noch sehr nahe beieinander liegen, wird dennoch das beste Ergebnis der Top-5 für die STS gewählt. So wird sichergestellt, dass der Merkmalsvektor die Ergebnisse der STS für die Klassifikation optimal nutzen kann.

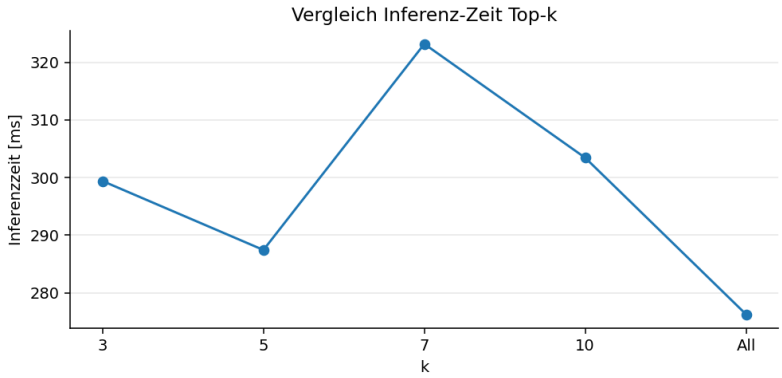

Abbildung 38: Auftragung der Inferenzzeit für Top-k mit k=[3, 5, 7, 10, alle]

5.3 Kernel-Funktion

Wie in Unterabschnitt 2.2.2 bereits ausgeführt, ist die Kernel-Funktion ein zentraler Bestandteil der SVM und ist verantwortlich für die Berechnung der Hyperebene im Vektorraum zur Trennung der Klassen. Da es keine Default-Lösung für bestimmte Klassifikations-Probleme gibt, wird die Art der Kernel-Funktion und deren Parameter experimentell bestimmt. Hier gilt die folgende Trainingskonfiguration aus Tabelle 11

Tabelle 11: Trainingskonfiguration der Kernel Analyse

Merkmalszusammensetzung	9feat
Trainingsdaten pro Klassen	120
Evaluierungsdaten pro Klasse	30
Top-k	k=5

Da es sich in dieser Arbeit um eine Text-Klassifikation handelt, ist es auf Basis der Erkenntnisse von Zhang *et al.* [41] naheliegend, einen linaren Kernel für eine bessere Performance zu verwenden. Lineare Kernels sind zudem Zeit- und Ressourcenschonender als komplexe Kernels und stellen allgemein in der Textklassifikation den

State-of-the-art dar. Im Zuge der folgenden Evaluierung wird diese Annahme in einer Gegenüberstellung mit verschiedenen nichtlinearen Kernel (RBF, Poly, Sigmoid) überprüft. Der Parameter C wird in diesem Zuge ebenfalls ermittelt. Python erleichtert diese Aufgabe, da es spezielle Funktionen zur Ermittlung der optimalen Kernel und Parameter gibt. Für jeden untersuchten Kernel wird eine Auswahl an Parametern p getestet, in diesem Fall $p \in \{0.1, 1, 10, 100\}$. Die Funktion *GridSearchCV*() analysiert alle Kombinationen der Parameter und verwendet eine, Cross-Validation zur Evaluierung der Ergebnisse. Der beste Parameterwert wird zurückgegeben. Die Ergebnisse sind in Abbildung 39 aufgetragen.

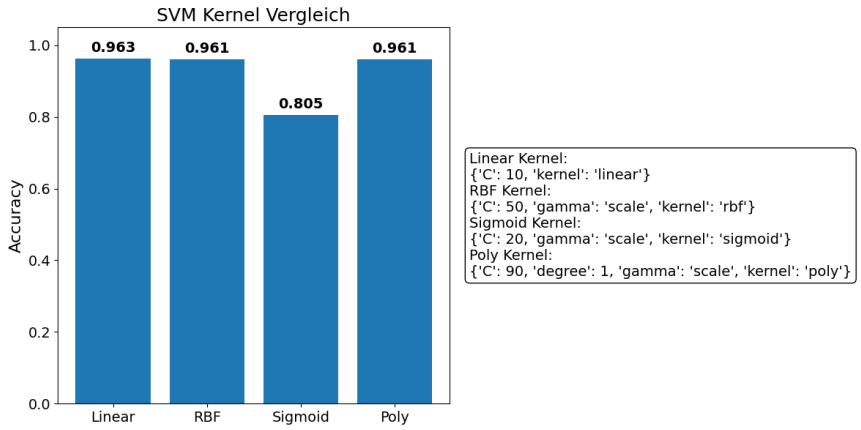

Abbildung 39: Auftragung der Cross-Validation Scores verschiedener Kernels mit der GridSearch-Ausgabe für den optimalen Parameter C

Die Kernels Linear, RBF und Poly haben einen sehr ähnlichen Score, wobei der lineare Kernel um 0,003 besser abschneidet, was einer Genauigkeit von 0,3 % entspricht. Die optimalen C-Werte sind in Abbildung 39 rechts neben der Auftragung dargestellt. Für den linearen Kernel wird C=10 als optimal ermittelt. Diese Zusammensetzung eines linearen Kernels mit dem Parameter C=10 wird für die weiteren Evaluierungen festgelegt.

5.4 Performance

Im Zuge der Performance-Analyse erfolgt die Betrachtung der Ausführungsgeschwindigkeit. Diese beinhaltet die Zeit, die es benötigt, um das SVM-Modell zu trainieren sowie die Inferenzzeit. Im Programm selbst werden die Zeitstempel unmittelbar vor und nach dem ausführenden Codeabschnitt gesetzt, sodass keine weiteren Berechnungen die Zeitnahme beeinflussen. Für einen direkten Vergleich werden die Zeitspannen mit verschiedenen Klassenzahlen $k \in \{4, 8, 12, 16, 20, 24, 28\}$ betrachtet, wobei 28 Klassen wegen den doppelten FTs 14 Bedienelementen entsprechen. Da im Zuge dieser Arbeit nicht alle möglichen Elemente des BMV einbezogen werden, soll eine Abschätzung der Zeitentwicklung für eine höhere Klassenzahl stattfinden, als die hier genutzte.

Für beide Evaluierungen wird die folgende Konfiguration aus Tabelle 12 verwendet. Pro Klasse werden 250 Datenpaare im Trainingsdatensatz genutzt (Aufsplittung 80 %-20 % für Training und Validierung).

Tabelle 12: Trainingskonfiguration der Performanceanalyse

Merkmalszusammensetzung	9feat
Kernel	linear, C=10
Trainingsdaten pro Klassen	200
Evaluierungsdaten pro Klasse	50
Top-k	k=5

5.4.1 Trainingszeit

Die Analyse umfasst die Betrachtung der Trainingszeit bei variabler Klassenmenge. In der Abbildung 40 ist der Kurvenverlauf in blau aufgetragen, welcher sich quadratisch steigend verhält. Mit den gegebenen Trainingsdaten von 250 pro Klasse, beträgt die Trainingszeit inklusive der Validierung mit 28 Klassen bereits 7,35 s. Durch eine quadratische Regression (siehe Abbildung 40) wird die Kurvengleichung

$$f_{train}(x) = 151,91x^2 - 32,06x + 101,37 \tag{51}$$

mit einer Güte von $R^2 = 0,9993$ ermittelt, in welcher $x \in \mathbb{Z}$ der Anzahl an Klassen entspricht. Die Trendlinie ist in orange in Abbildung 40 geplottet. Eine Schätzung der Trainingszeit für 100 und 200 Klassen, würde eine Dauer von

$$f_{train}(100) = 25,27\,min \tag{52}$$

$$f_{train}(200) = 101,17\,min \tag{53}$$

ergeben. Ein anpassendes Training des Modells würde somit je nach Klassenanzahl ca. 0,5 - 1,5 h Zeit in Anspruch nehmen und muss dementsprechend in den Tagesablauf eingeplant werden.

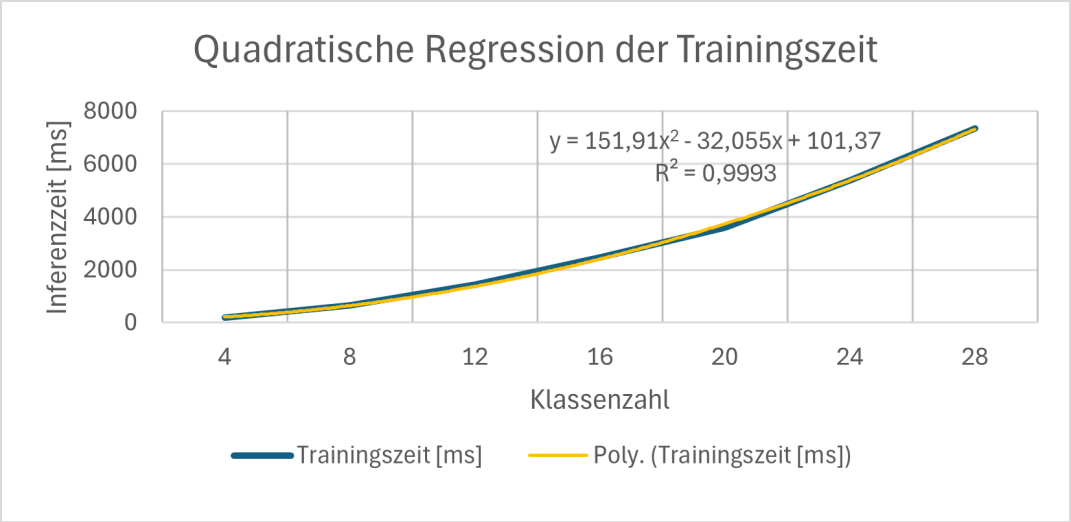

Abbildung 40: Auftragung der Trainingszeit bei verschiedener Anzahl an Klassen inklusive quadratischer Regression zur Formelbestimmung mit Excel

5.4.2 Inferenzzeit

Die Betrachtung der Inferenzzeit liefert einen annähernd linearen Verlauf (vergl. blaue Kurve in Abbildung 41), was zu dem genutzten linearen Kernel der SVM passt. Wie zuvor bei der Auftragung der Trainingszeit erfolgt die Bestimmung der Kurvengleichung über eine lineare Regression. Das Ergebnis ist die Gleichung

$$f_{inf}(x) = 0,14x + 488,22 \tag{54}$$

mit einer Güte von $R^2 = 10^{-5}$ ermittelt, in welcher $x \in \mathbb{Z}$ der Anzahl an Klassen entspricht. Die Trendlinie ist in orange in Abbildung 41 geplottet. Eine Schätzung der Inferenzzeit auf Basis dieser Gleichung ergibt

$$f_{inf}(100) = 501,72\,ms \tag{55}$$

$$f_{inf}(200) = 515,22\,ms. \tag{56}$$

Diese Werte zeigen keine Abweichungen zu den bestehenden Daten (eingetragen in Abbildung 41), was bei einer Geradensteigung von 0,14 zu erwarten ist (siehe auch Differenz zwischen den Ergebnissen von $f_{inf}(100)$ und $f_{inf}(200)$). Wird ein mittlerer Wert von ca. 500 ms als Inferenzzeit angenommen, stellt dies auch bei einer hohen Klassenzahl kein Problem dar.

Zusammenfassend befindet sich die Inferenzzeit in einem guten Rahmen und verlängert die Ausführungszeit des Programms marginal (oberer Millisekundenbereich). Die Trainingszeit nimmt eine abschätzbare Zeit in Anspruch, auf Basis dieser ein Retraining bei Bedarf geplant werden kann.

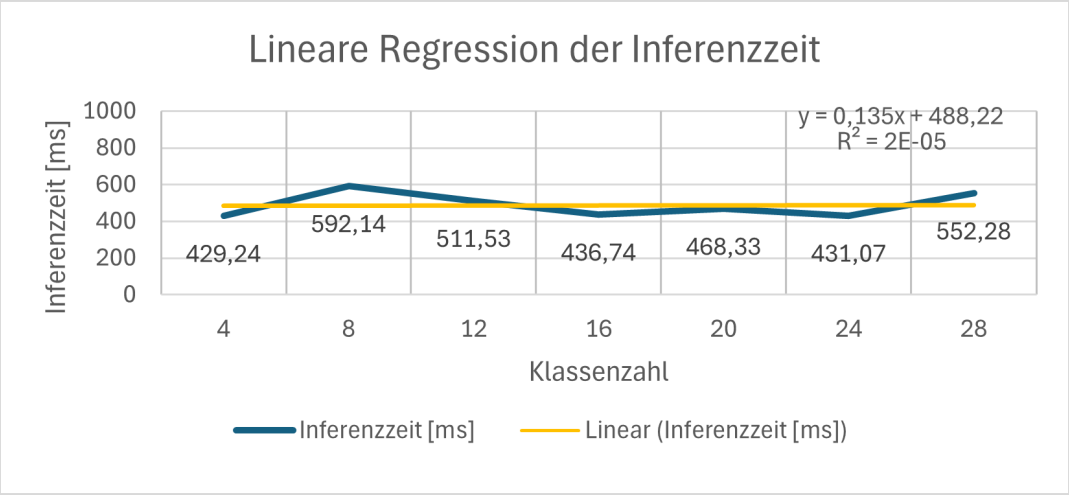

Abbildung 41: Auftragung der Inferenzzeit bei verschiedener Anzahl an Klassen inklusive quadratischer Regression zur Formelbestimmung mit Excel

5.5 Ressourcen

Dieses Kapitel analysiert den Ressourcenbedarf des Modells, was den Speicherbedarf und die CPU-Auslastung inkludiert. Wie zuvor bei der Performance in Unterabschnitt 5.4 werden die Messungen auch hier für eine unterschiedliche Anzahl an Klassen $k \in \{4, 8, 12, 16, 20, 24, 28\}$ vorgenommen, um aus deren Verlauf den Ressourcenbedarf für spätere höhere Klassenzahlen abzuschätzen. Für die jeweiligen Messungen erfolgt die selbe Konfiguration des Modells wie auch bei der Performance-Messung. Eine Übersicht ist in Tabelle 13 gegeben.

Tabelle 13: Trainingskonfiguration der Ressourcenanalyse

Merkmalszusammensetzung	9feat
Kernel	linear, C=10
Trainingsdaten pro Klassen	200
Evaluierungsdaten pro Klasse	50
Top-k	k=5

5.5.1 RAM-Nutzung

Da die Speichernutzung primär beim SVM-Training ins Gewicht fällt, wird im Folgenden der RAM nur während des Trainingsprozesses betrachtet. Die Messung der RAM-Speicher-Datenpunkte erfolgt unmittelbar nach denen der Zeitstempel. Die erhobene Größe entspricht dabei der Differenz des RAMs zwischen dem Anfang

und Ende des Trainings. Die Datenpunkte der verschiedenen Klassenanzahl zusammen mit einer linearen Regression zur Ermittlung der Kurvenfunktion sind in Abbildung 42 abgebildet. Die ermittelte Funktion ist

$$f_{RAM}(x) = 8,66x - 5,84 \tag{57}$$

mit einer Güte von $R^2 = 0,9956$. Entsprechend lässt sich der RAM-Bedarf für das Training von 100 und Klassen leicht abschätzen mit

$$f_{RAM}(100) = 860,16\ MB \tag{58}$$

$$f_{RAM}(200) = 1726,16\ MB = 1,73\ GB. \tag{59}$$

Bei einem installierten RAM von 16 GB, stellt diese Auslastung somit kein Problem dar. Erst bei ca. 1000 Klassen würde der RAM zu 50 % ausgelastet sein. Dennoch sollten nicht zu viele Anwendungen parallel ausgeführt werden, um die Ausführungszeit nicht unnötig zu verlängern.

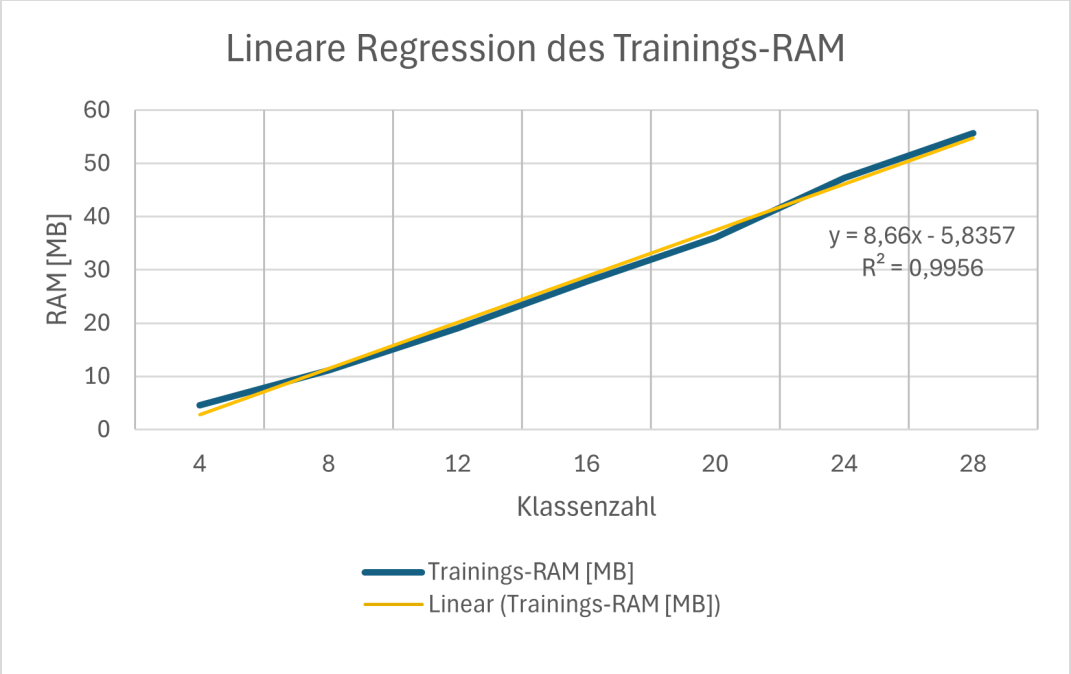

Abbildung 42: Auftragung des Trainings-RAM bei verschiedener Anzahl an Klassen inklusive linearer Regression zur Formelbestimmung mit Excel

5.5.2 CPU-Nutzung

Die Messung der CPU-Auslastung startet und endet jeweils mit einer halben Sekunde verzögert, um die Auslastung im Kontext des vorherrschenden Auslastungszustandes zu beurteilen. Auf der ausführenden Hardware ist das Training in der Zeit die einzige laufende Applikation. Die Aufzeichnung ist in folgender Abbildung 43 abgebildet, wobei der Prozess des Trainings in rot und die validierende Vorhersage in grün hinterlegt sind. Es wird hier der Verlauf der 28 Klassen (siehe Abbildung 43) sowie der acht Klassen (siehe Abbildung 44) betrachtet, um einen Vergleich bei variierender Klassenzahl zu haben.

Insgesamt verläuft der Trainingsprozess für 28 Klassen (siehe Abbildung 43) ohne Ausreißer bei ca. 10 % der CPU-Auslastung mit einer Varianz von ca. 5-10 %. Auch bei der Inferenz befindet sich die CPU-Auslastung in diesem Bereich, übersteigt jedoch auch öfters die 20 %. Während des Trainings und der Vorhersage sind mehrere Peaks bis 100 % vorhanden. Diese Peaks markieren mit hoher Wahrscheinlichkeit bestimmte Rechenoperationen, die einen hohen Berechnungsaufwand fordern. Diese beinhalten beispielsweise das Anlegen interner Strukturen, das Kopieren von Daten oder der Vorbereitung und Berechnung von Matrizen und Support-Vektoren. Da die Berechnung nur auf der CPU stattfinden können, bedeutet die Peaks zudem, dass die CPU an den entsprechenden Stellen optimal ausgenutzt wird.

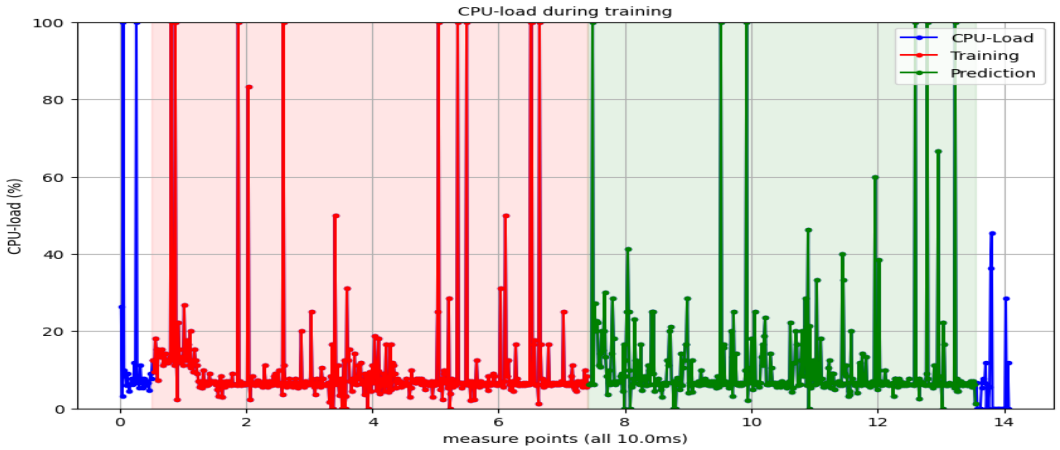

Abbildung 43: Aufzeichnung der CPU-Auslastung während des Trainings (rot) und der Validierung (grün) für 28 Klassen

Die vergleichende Betrachtung des Verlaufes für acht Klassen zeigt in Abbildung 44 ein homogeneres Bild wie zuvor für 28 Klassen. Die CPU-Auslastung schwankt ebenfalls um 10 %, besitzt jedoch nicht dieselbe Anzahl und Höhe der Peaks wie zuvor bei 28 Klassen. Dies ist ein starker Hinweis darauf, dass die acht Klassen einen deutlich geringeren Rechenaufwand fordern, als die 28. Dies ist eine logische Schlussfolgerung, da es bei einer steigenden Klassenzahl mehr Berechnungen erfordert, die Parameter zur Trennung im Vektorraum zu bestimmen. Es sind mehr Support-Vektoren vorhanden und die Klassen selbst sind weniger klar separierbar wie bei acht Klassen.

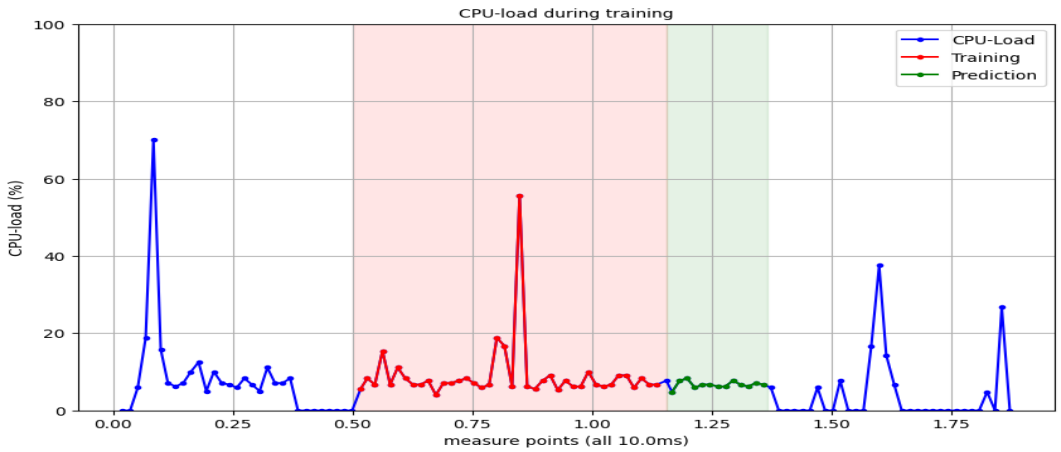

Abbildung 44: Aufzeichnung der CPU-Auslastung während des Trainings (rot) und der Validierung (grün) für 8 Klassen

Zusammenfassend stellt sowohl die RAM-Belegung als auch die CPU-Auslastung keine kritische Belastung für die Anwendung auf einem ressourcenlimitierten Laptop dar. Dies gilt sowohl für die aktuelle Analyse sowie nach aktueller Schätzung für zukünftige umfangreichere Modelle. Es muss jedoch beachtet werden, dass sowohl die RAM-Nutzung als auch die CPU-Auslastung abhängig von der Anzahl der eingebundenen Klassen sind. Dies sollte bei einer Erweiterung des Modells berücksichtigt werden.

5.6 Qualität

Das folgende Unterkapitel befasst sich ausführlich mit der Ermittlung aussagekräftiger Qualitäts-Informationen. Insbesondere erfolgt eine Untersuchung der Modell-Genauigkeit, erhoben durch die Methode der Cross-Validation. Da hier alle Daten in mehreren Iterationen sowohl zum Trainieren als auch zur Validierung verwendet werden, ist diese Methode für das Gesamtmodell aussagekräftiger, als der Genauigkeitswert bei nur einem Training. Zusätzlich werden die Größen der *Precision*, des *Recall* und *F1* betrachtet, die zusätzlich Aufschluss über die Verteilung der falschpositiven Ergebnisse liefern. Zuletzt wird der Trainingsverlauf betrachtet, der Aufschlüsse über die Qualität der Trainingsdaten gibt und wie gut das Modell im Laufe des Trainings generalisiert. Die verwendete Merkmalszusammensetzung ist wie bereits zuvor *9feat* mit 28 Klassen (*28class*) und unter der Einbeziehung der Top-5 Ergebnisse der STS. Im Zuge der Qualitäts-Analyse wird ein nochmals erweiterter Datensatz mit 250 Datenpaaren pro Klasse verwendet. Eine Übersicht ist in Tabelle 14 gegeben.

Tabelle 14: Trainingskonfiguration der Ressourcen Analyse

Merkmalszusammensetzung	9feat
Kernel	linear, C=10
Trainingsdaten pro Klassen	200
Evaluierungsdaten pro Klasse	50
Top-k	k=5

5.6.1 Genauigkeit

Für weitere, nähere Betrachtungen der Modell-Genauigkeit, werden neben der Cross-Validation (Gesamtscore 94,04 %) die Metriken *F1*, *Precision* und *Recall* der einzelnen Klassen betrachtet. Eine Übersicht ist in Abbildung 45 dargestellt.

Bei der Betrachtung aller Klassen im Vergleich sind leichte Schwankungen im *F1*-Score zwischen den Klassen erkennbar, jedoch befinden sich diese alle zwischen 88 %-100 % (Mittelwert 93,82 %, Standardabweichung 2,92 %). Es sind keine groben Ausreißer vorhanden. Diese Auftragung zeigt somit eine homogene Leistungsverteilung und spricht alle Klassen gleichermaßen mit wenigen natürlichen Schwankungen an. Bei zwei Klassen wird ein *F1*-Score von 100 % erreicht, wohingegen bei jeweils elf Klassen der *Recall* dominiert und bei dreizehn Klassen die *Precision*. Bei zwei Klassen liegen die drei Metriken bei demselben Wert. Eine Auflistung der Klassen und deren Ergebnis-Ausprägung sind in Tabelle 15 gesammelt aufgetragen.

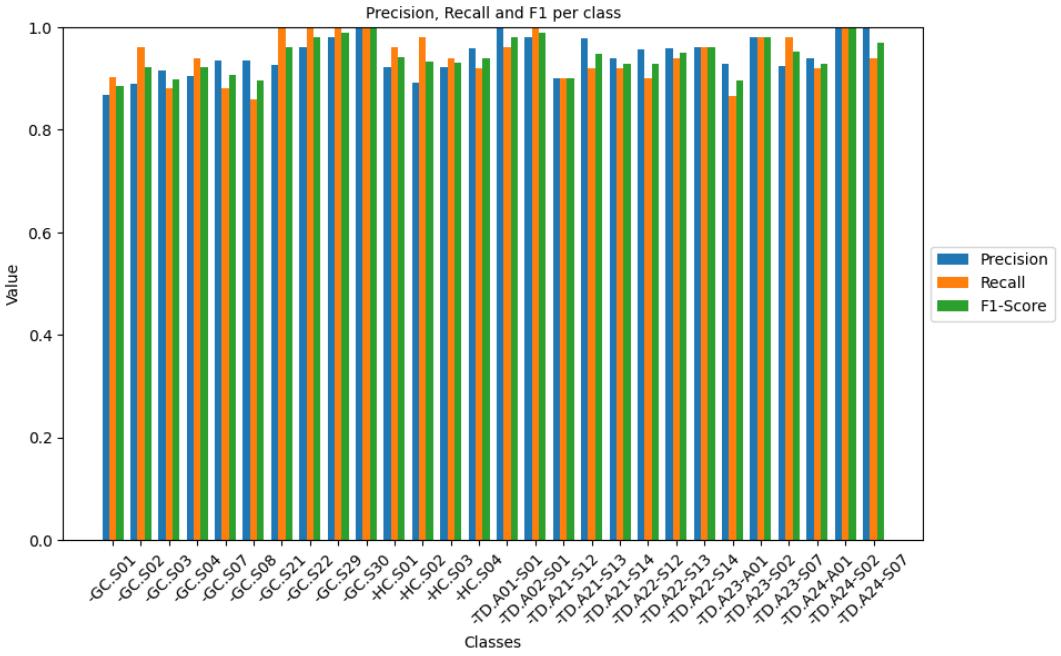

Abbildung 45: Übersicht des *F1*-Scores, der *Precision* und des *Recall*s für alle Klassen

Anhand der Auswertung lassen sich keine klaren Tendenzen zu einem vergleichsweise hohen Recall oder Precision ableiten. Die Differenzen zwischen den beiden Metriken betragen maximal 9 %, was natürlichen Schwankungen zugrunde liegt und in einem realen ML-Szenario in den Grenzen eines ausgeglichenen Verhaltens liegen. Das Modell neigt folglich weder dazu, zu konservativ zu sein (hohe Precision) noch zu einer hohen Risikobereitschaft (hoher Recall). Die Entscheidungsschwelle und die Margin lassen sich als gut austariert annehmen, ebenso spricht die gezeigte Verteilung für eine gute Ausgewogenheit der Klassen. Aus diesem Grund werden keine weiteren Maßnahmen zur Verbesserung erörtert.

Tabelle 15: Auflistung der Klassen und deren Verteilung der *Precision* und *Recall*: P = Precision, R = Recall

Verteilung	Bedienelement	Bezeichner (FTx)
100 % Scores	Lösen/Aktivieren der automatischen Bremse	-GC.S30 (FT2)
	Heben und Senken des Pantographen	-TD.A24-S02 (FT2)
P = R	Auswahl der Fahrtrichtung "vorwärts"	-TD.A21-S12 (FT1)
	Auswahl der Fahrtrichtung "rückwärts"	-TD.A22-S14 (FT2)
P > R	Entbesetzen des Führerstandes	-GC.S03 (FT1)
	Bedienung des Notbrems-Knopf	-GC.S07/S08
	Lok-Batterie ausschalten	-HC.S04 (FT2)
	Bedienung des Sicherheits-Fahrschalter-Pedals	-TD.A01-S01 (FT1)
	Auswahl der Fahrtrichtung "neutral"	-TD.A21-S13 (FT1)
	Auswahl der Fahrtrichtung "rückwärts"	-TD.A21-S14 (FT1)
	Auswahl der Fahrtrichtung "vorwärts"	-TD.A22-S12 (FT2)
	Auswahl der Fahrtrichtung "neutral"	-TD.A22-S13 (FT2)
	Bedienung des Fahr-Brems-Hebels (TBC)	-TD.A23-A01 (FT1)
	Heben und Senken des Pantographen	-TD.A23-S02 (FT1)
	Bedienung des Fahr-Brems-Hebels (TBC)	-TD.A24-A01 (FT2)
	Bedienung des Hauptschalter-Hebels	-TD.A24-S07 (FT2)
R > P	Besetzen des Führerstandes	-GC.S01 (FT1)
	Besetzen des Führerstandes	-GC.S02 (FT2)
	Entbesetzen des Führerstandes	-GC.S04 (FT2)
	Lösen der Parkbremse	-GC.S21/S22
	Lösen/Aktivieren der automatischen Bremse	-GC.S29 (FT1)
	Lok-Batterie anschalten	-HC.S01/S02
	Lok-Batterie ausschalten	-HC.S03 (FT1)
	Bedienung des Sicherheits-Fahrschalter-Pedals	-TD.A02-S01 (FT2)
	Bedienung des Hauptschalter-Hebels	-TD.A23-S07 (FT1)

5.6.2 Trainingsverlauf

Die Lernkurve des Modells lässt viele Schlüsse über die Qualität der Trainingsdaten und deren Nutzen für die Generalisierungsfähigkeit zu. Zudem lässt sich bereits anhand des Kurvenverlaufs mögliche Tendenzen zum *Under-* oder *Überanpassung* erkennen und inwieweit das Modell mit den vorhandenen Daten in seinem Potential ausgeschöpft ist. Der Verlauf in Abbildung 46 stellt die Genauigkeit des Modells in Abhängigkeit der verwendeten Trainingsdatenmenge dar. Dabei wird zwischen der Genauigkeit der Trainings- und Validierungsdaten unterschieden.

Der Verlauf der Genauigkeit der Trainingsdaten liegt nahezu konstant bei ca. 98 %. Das lässt darauf schließen, dass mit bereits sehr wenigen Daten eine Hyperebene berechnet wird, welche die Daten nahezu perfekt trennt und auch bei mehr Daten stabil bleibt. Die Merkmale liegen entsprechend so im Vektorraum, dass bereits eine einfache lineare Trennfläche ausreicht. Einen Beitrag zur frühen Stabilität leistet der Parameter C, welcher in diesem Fall mit C = 10 hoch angesetzt wurde. Ein hohes C bedeutet, dass das Modell eine möglichst korrekte Trennung der Trainingsdaten anstrebt. Dies führt jedoch auch zur engen Anpassung an die Trainingsdaten, was zu einer künstlichen Anhebung der Trainings-Genauigkeit führt und auf Überanpassung hindeuten kann. Ausreißer, die nicht korrekt klassifiziert werden, sind entsprechend untypische oder schwierige Fälle und lassen sich auch mit mehr Trainingsdaten nicht korrigieren.

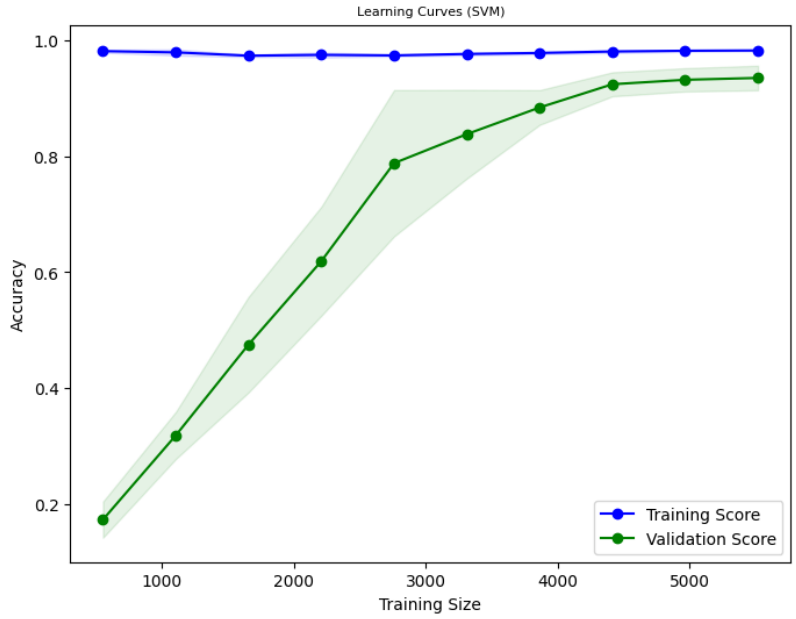

Abbildung 46: Auftragung der Lernkurve des SVM Trainings mit Abbildung der Genauigkeit der Validierungs- und Trainingsdaten gegen die Trainingsdatenmenge

Die Genauigkeit der Validierungsdaten verhält sich konvergierend zur Kurve der Trainingsdaten. Jeder zusätzliche Datensatz stabilisiert die lineare Entscheidungsgrenze und verbessert somit die Generalisierung. Ab ca. 3000 Trainingsdaten flacht die Kurve ab und nährt sich ab ca. 4000 Trainingsdaten einem Plateau bei ungefähr 93 %. Die 4000 Daten entsprechen dabei aufgerundet 180 Datenpaaren pro Klasse. Das Erreichen des Plateaus bedeutet, dass das Maximum der generalisierbaren Leistung erreicht ist. Die weiterhin bestehende Distanz zur Kurve der Trainingsdaten bedeutet eine geringfügige Rest-Varianz. Das Modell generalisiert gut, jedoch bleibt ein geringes Überanpassung, welches nicht jede Feinheit in den Merkmalen trennen

kann. Die Varianz kann auch durch leicht überlappende Klassen hervorgerufen sein.

Anhand der Grafik ist ersichtlich, dass auch bei genügend Trainingsdaten ein leichtes Überanpassung besteht. Da der Parameter C stringent gewählt ist, würde eine Verminderung des Wertes eine größere Toleranz gegenüber Trainingsfehlern ermöglichen. Das sorgt für eine Vergrößerung des Raumes zwischen den Support Vektoren (Margin) und würde zu einer Stabilisierung des Modells inklusive eines geringeren Überanpassung beitragen.

5.7 Zusammenfassung

Das letzte Kapitel widmete sich der umfassenden Evaluierung des Zuordnungssystems, dessen Ergebnisse in Tabelle 16 tabellarisch zusammengefasst sind. Die eingesetzten Methoden umfassten die Cross-Validation, Confusion-Matrizen, Lernkurven sowie Messungen zur Performance, dem Ressourcenbedarf und der Modellqualität. Ein zentraler Bestandteil der Evaluierung war die Untersuchung verschiedener Merkmalszusammensetzungen. Dabei zeigte sich, dass das Eingabe-Embedding den größten Beitrag zur Modellleistung liefert, während STS-basierte Merkmale die Robustheit erhöhen, jedoch nur einen geringen Einfluss auf die Genauigkeit haben. Ergänzend wurde die optimale Top-k Auswahl für die STS-Einbindung identifiziert mit $k = 5$ und einem linearen Kernel mit einem SVM-Parameter von $C = 10$. Die Performanceanalyse zeigte eine lineare beziehungsweise quadratische Entwicklung der Inferenz- und Trainingsdauer, die im Rahmen der gegebenen Hardwareanforderungen unkritisch bleiben. Auch die RAM- und CPU-Auslastung bewegten sich in einem akzeptablen Bereich für den Einsatz auf ressourcenlimitierten Geräten. Die Qualitätsanalyse ergab eine Cross-Validation-Genauigkeit von ca. 94 %, stabile F1-Werte über alle Klassen und eine Lernkurve, die ab etwa 180 Trainingspaaren pro Klasse ein Plateau erreicht. Damit bestätigte die Evaluierung, dass das entwickelte System unter den gegeben Einschränkungen zuverlässig arbeitet und für die Anwendung im Kontext der Validierungstest in der funktionalen Sicherheit geeignet ist.

Tabelle 16: Zusammenfassung der Evaluierungsergebnisse

Evaluierung	Eval.-Objekt	Ergebnis
Merkmale	Merkmale	*9feat*-Konfiguration; Textembedding als wichtigstes Merkmal; Weitere Analysen empfohlen
	STS-Ergebnisse	Top-k Ergebnisse der STS mit $k = 5$
Kernel	Kernelfunktion	Linearer Kernel mit $C = 10$
Performance	Trainingszeit	Quadratische Regression zur Abschätzung; Retraining mit >150 Klassen benötigt >1h; Entsprechende Planung erforderlich
	Inferenzzeit	Lineare Regression zur Abschätzung; Verlauf besitzt marginale Steigung von ca. 14 ms pro 100 Klassen; Inferenzzeit schwankt nahezu konstant um ca. 500 ms
Ressourcen	RAM-Nutzung	Lineare Regression zur Abschätzung; Verlauf besitzt marginale Steigung von ca. 900 MB pro 100 Klassen; Auslastung von <1000 Klassen unkritisch bei RAM von 16 GB
	CPU-Nutzung	Kontinuierliche Auslastung zwischen 10-20 % mit gelegentlichen Peaks
Qualität	Genauigkeit	Cross-Validation-Score: $CV = 94,04\%$; F1-Score: $F1 \in [88, 100]\%$; Größte Differenz zwischen Precision und Recall: $D_{max} = 9\%$; Gleichmäßige Schwankungen ohne Ausreißer, ausgeglichenes Verhalten zwischen den Klassen
	Train-Verlauf	Ab ca. 180 Trainingsdaten pro Klasse erreicht Validierungskurve ein Plateau bei ca. 93 %; Maximum der generalisierbaren Leistung erreicht; Rest-Varianz von ca. 5 % zur Trainingskurve bedeutet geringe Überanpassung;

6 Zusammenfassung und Ausblick

Zusammenfassung Die vorliegende Arbeit befasst sich mit der zentralen Problemstellung, dass im aktuellen Validierungsprozess sicherheitskritischer Lokomotivsysteme kein Verfahren bekannt ist, welches unter beschränkter Rechenleistung natürlichsprachliche Testanweisungen zuverlässig und automatisiert zugehörigen Hardware-Bedienelementen zuordnet. Die daraus abgeleiteten Forschungsfragen zielen auf die Erstellung eines entsprechenden Verfahrens unter Verwendung von ML (siehe FF1) und der Gestaltung eines Systems, welches ein solches Verfahren in einen bestehenden Prototypen integriert (siehe FF2).

Im Zuge dieser Arbeit wurde zur Erfüllung der FF1 ein hybrider Ansatz entwickelt, welcher die Stärken semantischer Transformer-Modelle mit den Vorteilen klassischer ML-Klassifikation kombiniert. Durch die Analyse geeigneter Modelle unter den gegebenen Hardwareeinschränkungen zeigte sich, dass der Sentence-Transformer *all-MiniLM-L6-v2* ein geeignetes Gleichgewicht zwischen semantischer Repräsentation und Ressourcenbedarf bietet. Die theoretischen Grundlagen des natürlichen Sprachverarbeitung, der semantischen Textähnlichkeit und kontextabhängiger Embeddings wurden zu diesem Zweck auf die in dieser Arbeit spezifische Testdomäne übertragen. Auf Basis dessen wurde eine SVM als Klassifikator gewählt, da sie auch mit kleinen Datenmengen performant arbeitet und eine klare Trennung im Merkmalsraum vornehmen kann. Um die Datenbank des BMV für die Klassifikation als Wissensbasis nutzbar zu machen, wurde auf deren Basis eine Vektordatenbank aufgebaut. Diese ermöglicht es, die Testschritte mit BMV-Informationen abzugleichen und daraus Merkmale für die Klassifikation abzuleiten. Der erzeugte Merkmalsvektor kombiniert das Testschritt-Embedding dabei mit statischen und relationalen STS-Metriken, wodurch robuste Klassifikationsentscheidungen auch bei Formulierungsvariationen und Synonymen möglich werden. Ein zusätzlicher Konfidenzmechanismus sowie eine Human-in-the-Loop-Kontrollinstanz stellen dabei eine Reduzierung kritischer Fehlzuordnungen sicher und stellen gleichzeitig eine Möglichkeit zur Verbesserung des Klassifikationssystems zur Verfügung.

Zur Beantwortung der FF2 wurde eine modulare Systemarchitektur entworfen und umgesetzt. Diese sieht eine klare funktionale Trennung zwischen Datenvorbereitung, Erzeugung und Bearbeitung der Vektordatenbank, Training der SVM und der Vorhersage der Bedienelemente vor. Die Implementierung in Python mit einer Konfigurationssteuerung erlaubt die Auswahl der gewünschten Funktionalitäten über eine minimalisierte Schnittstellennutzung. Über eine virtuelle Umgebung lässt sich das System nahtlos in bestehende Prototypen einbinden, ohne diese abgesehen von der Integration des Interfaces zu verändern. Gleichzeitig ermöglicht die Struktur eine spätere Erweiterung, beispielsweise einen Austausch des Sprachmodells oder die Erweiterung der Vektordatenbank um weitere Bedienelemente.

Die Evaluierung bestätigt die Wirksamkeit des aufgebauten Systems sowohl im Hinblick auf die Genauigkeit als auch auf den Ressourcenbedarf. Mit einer Cross-Validation von 94 % und einem stabilen F1-Wert über alle 28 Klassen hinweg erfüllt das Modell die Zielsetzung einer zuverlässigen Zuordnung der Testschritte zu den Bedienelementen unter begrenzten Trainingsdaten und Ressourcenverfügbarkeit. Die Performanceanalyse zeigt dabei, dass die Trainings- und Inferenzzeit mit den vorhandenen Hardwareeinschränkungen kompatibel sind und die RAM- und CPU-Belastungen im unkritischen Bereich liegen. Die Analyse der Lernkurve bestätigt zudem, dass die genutzte Trainingsdatenmenge von über 180 Datenpaaren für eine stabile Generalisierung ausreichend ist und somit die zur Verfügung stehenden Daten effektiv nutzt. Gleichzeitig zeigt die Analyse der Merkmalszusammensetzung, dass das Embedding der Testschritt-Eingabe den größten Beitrag für eine erfolgreiche Klassifikation liefert, während die STS-basierten Merkmale die Robustheit unterstützen.

Zusammenfassend konnte die in der Einleitung adressierte Problemstellung vollständig bearbeitet werden. Es wurde ein Verfahren entwickelt, welches unter den gegebenen Ressourcenbeschränkungen zuverlässig natürlichsprachliche Testanweisungen klassifiziert und dessen Architektur so gestaltet ist, dass eine flexible Integration und eine zukünftige Modifikationen gewährleistet sind. Das Projekt inklusive dessen Codedateien und Evaluationsdaten ist über ein Git-Projekt [3] zugänglich.

Ausblick Die Ergebnisse dieser Arbeit bilden die Grundlage für eine nachhaltige Weiterentwicklung der automatisierten Zuordnung von Testanweisungen im Bereich der funktionalen Sicherheit. Ein naheliegender weiterer Schritt besteht in der Erweiterung des Modells auf den vollständigen Umfang der BMV-Bedienelemente. Die durchgeführte Abschätzung der Ressourcennutzung zeigt, dass der Einsatz des Systems auch bei steigender Klassenzahl technisch realisierbar bleibt. Diese Abschätzung lässt sich jedoch nur durch eine praktische Überprüfung verifizieren.

Abgesehen davon bietet die Merkmalsanalyse Hinweise auf ein zukünftiges Optimierungspotenzial. Auch wenn die STS-Merkmale die Robustheit stützen, deutet der geringe Beitrag zur Verbesserung der Genauigkeit darauf hin, dass deren Beitrag und eine mögliche Substitution durch effizientere Merkmale untersucht werden sollte. Dies könnte die allgemeine Genauigkeit des Modells weiter verbessern.

Langfristig besteht die Möglichkeit, das entwickelte System über die Zuordnung von Bedienelementen hinaus einzusetzen. Beispielsweise gibt es bereits in anderen Testbereichen Testskripte, welche manuell eingesetzte E3-Bezeichnungen enthalten. Mithilfe der ML-Klassifikation ließen sich fehlerhaft zugeordne-

te Bedienelemente erkennen und korrigieren. Dies würde die Testqualität auch in anderen Bereichen erhöhen und so zeitliche Ressourcen und somit Kosten sparen.

Insgesamt zeigt die vorliegende Arbeit, dass die Nutzung von semantischen Embeddings und ML-Klassifikation einen leistungsfähigen und ressourceneffizienten Ansatz darstellt, um den manuellen Aufwand in sicherheitskritischen Validierungstests zu reduzieren. Die entwickelten Konzepte und Evaluierungsergebnisse schaffen eine solide Grundlage für zukünftige Weiterentwicklungen. Diese können eine umfangreiche Automatisierung und Qualitätssicherung im Testumfeld ermöglichen.

Literatur

[1] Welcome to Python.org. https://www.python.org/, December 2025.

[2] Herve Abdi, Dominique Valentin, and Betty Edelman. *Neural networks*. Number 124. Sage, 1999.

[3] Wiebke Albers. control_element_classification. https://github.com/Wiebkea214/control_element_classification, 2026. MIT License.

[4] Allianz pro Schiene. Verkehrssicherheit im vergleich: Bahn ist sicherstes verkehrsmittel. https://www.allianz-pro-schiene.de/themen/verkehrssicherheit/, 2024. Zehnjahresvergleich 2014–2023: Bahn 52-mal sicherer als Auto, 137-mal weniger schwere Verletzungen.

[5] Amer F. A. H. Alnuaimi and Tasnim H. K. Albaldawi. An overview of machine learning classification techniques. *BIO Web of Conferences*, 97:00133, 2024. Publisher: EDP Sciences.

[6] Ethem Alpaydin. *Introduction to Machine Learning, fourth edition*. MIT Press, March 2020.

[7] Elham Babaee, Nor Badrul Anuar, Ainuddin Wahid Abdul Wahab, Shahaboddin Shamshirband, and Anthony T. Chronopoulos. An overview of audio event detection methods from feature extraction to classification. *Applied Artificial Intelligence*, 31(9-10):661–714, 2017.

[8] Ron Bell. Introduction and Revision of IEC 61508. In Chris Dale and Tom Anderson, editors, *Advances in Systems Safety*, pages 273–291, London, 2011. Springer.

[9] Jens Braband. Funktionale Sicherheit. In Jochen Holzfeind, Jia Liu, and Ferdinand Pospischil, editors, *Handbuch Eisenbahninfrastruktur*, pages 559–613. Springer, Berlin, Heidelberg, 2025.

[10] Dhivya Chandrasekaran and Vijay Mago. Evolution of semantic similarity—a survey. *ACM Comput. Surv.*, 54(2), February 2021.

[11] K. R. Chowdhary. Natural Language Processing. In *Fundamentals of Artificial Intelligence*, pages 603–649. Springer India, New Delhi, 2020.

[12] Cim-Team Latin Market. Electrial design software - e3.series. https://www.e3seriescenters.com/en/. Zugriff am 16. März 2026.

[13] Christiane Fellbaum, editor. *WordNet: An Electronic Lexical Database*. MIT Press, Cambridge, MA, 1998.

[14] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. Retrieval-Augmented Generation for Large Language Models: A Survey. http://arxiv.org/abs/2312.10997, March 2024. arXiv:2312.10997 [cs].

[15] Margherita Grandini, Enrico Bagli, and Giorgio Visani. Metrics for multi-class classification: an overview. https://arxiv.org/abs/2008.05756, 2020.

[16] Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. Minillm: Knowledge distillation of large language models. https://arxiv.org/abs/2306.08543, 2024.

[17] Soroush Heydari and Qusay H. Mahmoud. Tiny Machine Learning and On-Device Inference: A Survey of Applications, Challenges, and Future Directions. *Sensors*, 25(10):3191, January 2025.

[18] John Hunt. *Python Virtual Environments*, pages 469–486. Springer International Publishing, Cham, 2023.

[19] Maximilian Hübner. Teilautomatisierung von fusi-tests in der traxx3 train0-testumgebung. Masterarbeit, Hochschule Mannheim, Fakultät Informationstechnik, September 2024. Erstprüfer: Prof. Dr. Jens-Matthias Bohli. Zweitprüfer: M. Sc. Jonathan Rein.

[20] Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition, with Language Models*. 3rd edition, 2026. Online manuscript released January 6, 2026.

[21] Irina Klyueva. Improving quality of the multiclass svm classification based on the feature engineering. In *2019 1st International Conference on Control Systems, Mathematical Modelling, Automation and Energy Efficiency (SUMMA)*, pages 491–494, 2019.

[22] Petre Lameski, Eftim Zdravevski, Riste Mingov, and Andrea Kulakov. Svm parameter tuning with grid search and its impact on reduction of model overfitting. In Yiyu Yao, Qinghua Hu, Hong Yu, and Jerzy W. Grzymala-Busse, editors, *Rough Sets, Fuzzy Sets, Data Mining, and Granular Computing*, pages 464–474, Cham, 2015. Springer International Publishing.

[23] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474. Curran Associates, Inc., 2020.

[24] Goutam Majumder, Partha Pakray, Alexander Gelbukh, David Pinto, Goutam Majumder, Partha Pakray, Alexander Gelbukh, and David Pinto. Semantic Textual Similarity Methods, Tools, and Applications: A Survey. *Computación y Sistemas*, 20(4):647–665, December 2016.

[25] Sabrina J. Mielke, Zaid Alyafeai, Elizabeth Salesky, Colin Raffel, Manan Dey, Matthias Gallé, Arun Raja, Chenglei Si, Wilson Y. Lee, Benoît Sagot, and Samson Tan. Between words and characters: A brief history of open-vocabulary modeling and tokenization in nlp. https://arxiv.org/abs/2112.10508, 2021.

[26] George A. Miller. Wordnet: A lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[27] Saif M. Mohammad and Graeme Hirst. Distributional measures of semantic distance: A survey. *arXiv preprint arXiv:1203.1858*, 2012.

[28] Felix Mohr and Jan N. van Rijn. Learning curves for decision making in supervised machine learning: a survey. *Machine Learning*, 113:8371–8425, 2024.

[29] National Instruments. Was ist teststand? https://www.ni.com/de/shop/electronic-test-instrumentation/application-software-for-electronic-test-and-instrumentation-category/what-is-teststand.html, 2025. Zugriff am 14. Juli 2025.

[30] Jay F. Nunamaker. Systems development in information systems research. *Journal of Management Information Systems*, 7(3):89–106, 1991.

[31] Bohdan Pavlyshenko and Mykola Stasiuk. SEMANTIC SIMILARITY ANALYSIS USING TRANSFORMER-BASED SENTENCE EMBEDDINGS. *Electronics and information technologies /* , (30):43–58, August 2025.

[32] Malgorzata Pikies and Junade Ali. Analysis and safety engineering of fuzzy string matching algorithms. *ISA Transactions*, 113:1–8, 2021.

[33] Nils Reimers. Minilm-l6-h384-uncased · hugging face. https://huggingface.co/nreimers/MiniLM-L6-H384-uncased.

[34] Nils Reimers and Iryna Gurevych. Sentence-bert: all-minilm-l6-v2. https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2, 2021. Licensed under Apache License Version 2.0.

[35] S. Selva Birunda and R. Kanniga Devi. A Review on Word Embedding Techniques for Text Classification. In Jennifer S. Raj, Abdullah M. Iliyasu, Robert Bestak, and Zubair A. Baig, editors, *Innovative Data Communication Technologies and Application*, pages 267–281, Singapore, 2021. Springer Singapore.

[36] S. Selva Birunda and R. Kanniga Devi. A Review on Word Embedding Techniques for Text Classification. In Jennifer S. Raj, Abdullah M. Iliyasu, Robert Bestak, and Zubair A. Baig, editors, *Innovative Data Communication Technologies and Application*, pages 267–281, Singapore, 2021. Springer Singapore.

[37] Michael Springer. *KI →ML*, pages 335–337. Springer Berlin Heidelberg, Berlin, Heidelberg, 2019.

[38] Richard S. Sutton and Andrew Barto. *Reinforcement learning: an introduction*. Adaptive computation and machine learning. The MIT Press, Cambridge, Massachusetts, nachdruck edition, 1998.

[39] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[40] Li Yujian and Liu Bo. A normalized levenshtein distance metric. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(6):1091–1095, 2007.

[41] Wen Zhang, Taketoshi Yoshida, and Xijin Tang. Text classification based on multi-word with support vector machine. *Knowledge-Based Systems*, 21(8):879–886, 2008.

[42] Zhi-Hua Zhou. *Empirical Error and Overfitting*, chapter 2.1. Springer, 2021.

[43] Zhi-Hua Zhou. *Evaluation Methods*, chapter 2.2. Springer, 2021.

[44] Zhi-Hua Zhou. *Linear Models*, chapter 3. Springer, 2021.

[45] Zhi-Hua Zhou. *Support Vector Machine*, chapter 6. Springer, 2021.