Szenariobasierte Validierung selbstlernender Systeme mit individueller Benutzeraktion durch metamorphe Beziehungen

Zur Erlangung des akademischen Grades eines

DOKTORS DER INGENIEURWISSENSCHAFTEN
(Dr.-Ing.)

von der KIT-Fakultät für
Elektrotechnik und Informationstechnik
des Karlsruher Instituts für Technologie (KIT)

angenommene

DISSERTATION

von

M.Sc. Marco Stang
geb. in Bad-Mergentheim

Tag der mündlichen Prüfung:	15.05.2025
Hauptreferent:	Prof. Dr.-Ing. Eric Sax
Korreferent:	Prof. Dr. rer. nat. Cornelius Neumann

Kurzfassung

Die Integration von Künstlicher Intelligenz (KI) in unseren Alltag schreitet unaufhaltsam voran. Die kontinuierliche Weiterentwicklung und enge Zusammenarbeit zwischen Mensch und KI-Systemen macht vertrauenswürdige Ergebnisse dieser Systeme unerlässlich. Aktuelle Validierungsmethoden für KI-Systeme sind unzureichend, da sie auf traditionellen Testfällen und Testorakeln basieren, die für selbstlernende Systeme schwer zu spezifizieren sind. Besonders herausfordernd ist die Erstellung von Testorakeln für Systeme mit individueller Benutzerinteraktion aufgrund der spezifischen Verhaltensweisen der Nutzer.

Der Stand der Technik umfasst zunächst die Klassifikation von Testorakeln in der Softwareentwicklung. Darauf aufbauend werden die Methoden zur Umsetzung dieser Testorakel analysiert. Die Analyse zeigt, dass das Metamorphe Testen eine geeignete Methode zur Lösung des Testorakelproblems darstellt und somit zur Validierung selbstlernender Systeme mit Benutzeraktionen beiträgt.

Basierend auf dem Stand der Technik wurde ein Konzept entwickelt, das die Verwendung von metamorphem Testen zur Validierung selbstlernender Systeme mit Benutzeraktionen einschließt. Das Konzept beginnt mit der Spezifikation relevanter Szenarien zur Überprüfung der Funktionalität selbstlernender Systeme. Zunächst wird eine Anforderungsanalyse durchgeführt, um die für das System relevanten Szenarien zu identifizieren und in funktionale Szenarien umzuwandeln. Diese logischen Szenarien werden entweder durch Expertenwissen oder mithilfe einer morphologischen Matrix strukturiert, was die Grundlage für die Szenarioerzeugung bildet. Im zweiten Bereich, der Szenarioerzeugung, werden die zuvor definierten logischen Szenarien in konkrete Szenarien umgewandelt. Eine zentrale Herausforderung ist hierbei die Erstellung realistischer Testdaten. Das Tool CAGEN (Context and Action Generation) wurde entwickelt, um modular Testdatensätze nach dem Providerprinzip zu erzeugen. CAGEN generiert Kontextdaten wie GPS-Koordinaten, Temperaturdaten und

Benutzeraktionen, die spezifisch auf den jeweiligen Kontext abgestimmt sind, und ermöglicht so die Erzeugung realistischer Testdaten für Ausgangstestfälle. Die Ausgangstestfälle sind die Grundlage für das Metamorphe Testen. Metamorphe Tests überprüfen bestimmte Eigenschaften des Systems under Test anhand von zwei oder mehr Folgetestfällen. Ausgangstestfälle und Folgetestfälle stehen dabei in einer metamorphen Beziehung zueinander. Zur Definition geeigneter metamorpher Beziehungen wurde ein dreistufiges Ebenenmodell entwickelt. Basierend auf diesem Modell wurden acht metamorphe Beziehungen zur Validierung einer selbstlernenden Komfortfunktion identifiziert.

Die Evaluation des Konzepts umfasst die Überprüfung der Einhaltung der definierten metamorpher Beziehungen anhand von drei Szenarien für drei selbstlernende Systeme zur Erkennung von Benutzeraktionen. Zur Auswertung der Testergebnisse der mehrdimensionalen Testdatensätze wurde eine Clustervisualisierung entwickelt, die mittels Hauptkomponentenanalyse (PCA) eine zweidimensionale Abbildung erzeugt und somit den Vergleich der Testergebnisse ermöglicht.

Abstract

The integration of Artificial Intelligence (AI) into our daily lives is progressing inexorably. The advancing development and collaboration between humans and AI systems require trustworthy results from these systems.

Current validation methods for AI systems are inadequate as they rely on traditional test cases and test oracles, which are difficult to specify for self-learning systems. Particularly challenging is the creation of test oracles for systems with individual user interaction due to the diverse and specific behaviors of users. The state of the art initially includes the classification of test oracles in software development. Building on this, the methods for implementing these test oracles are analyzed. The analysis shows that metamorphic testing is a suitable method for solving the test oracle problem and thus contributes to the validation of self-learning systems with user actions.

Based on the state of the art, a concept was developed that includes the use of metamorphic testing for the validation of self-learning systems with user actions. The concept begins with the specification of relevant scenarios to verify the functionality of self-learning systems. Initially, a requirements analysis is conducted to identify the scenarios relevant to the system and convert them into functional scenarios. These logical scenarios are structured either through expert knowledge or with the help of a morphological matrix, forming the basis for scenario generation. In the second phase, scenario generation, the previously defined logical scenarios are transformed into concrete scenarios. A central challenge here is the creation of realistic test data. The tool CAGEN (Context and Action Generation) was developed to modularly generate test datasets following the provider principle. CAGEN generates context data such as GPS coordinates, temperature data, and user actions, which are specifically tailored to the respective context, thereby enabling the creation of realistic test data for initial test cases. These initial test cases form the basis for metamorphic testing. Metamorphic tests verify certain properties of the system under test using two or more follow-up test cases. Initial test cases and follow-up test

cases are in a metamorphic relationship with each other. To define suitable metamorphic relationships, a three-tier model was developed. Based on this model, eight metamorphic relationships were identified for the validation of a self-learning comfort function.

The evaluation of the concept includes verifying compliance with the defined metamorphic relationships using three scenarios for three self-learning systems for detecting user actions. For the evaluation of the test results from the multidimensional test datasets, a cluster visualization was developed that uses Principal Component Analysis (PCA) to generate a two-dimensional representation, thereby enabling the comparison of the test results.

Danksagung

Mein besonderer Dank gilt Herrn Prof. Dr.-Ing. Eric Sax, der mich als Doktorvater mit Expertise, stetiger Unterstützung und vertrauensvollem Austausch durch die gesamte Promotionszeit begleitet hat. Seine konstruktiven Rückmeldungen und sein kritischer Blick haben die Arbeit maßgeblich geprägt. Cornelius danke ich herzlich für die Übernahme des Korreferats und die damit verbundene Unterstützung während der Schlussphase der Dissertation.

Ein besonderer Dank gilt meinen Kolleginnen und Kollegen am Institut für Technik der Informationsverarbeitung (ITIV) für die inspirierende Zusammenarbeit, den fachlichen Austausch und das angenehme Arbeitsklima.David und Daniel (Grimbo) danke ich für die inhaltliche Auseinandersetzung mit der Arbeit, ihre kritischen Fragen und hilfreichen Impulse. Martin (Böhme), Marc, Max und Veljko haben mit ihrer humorvollen Art und der richtigen Portion Ablenkung wesentlich zur Balance während der Arbeitszeit beigetragen – auch dafür ein herzliches Dankeschön.

Mein tief empfundener Dank gilt außerdem meiner Familie – Edel, Werner und meiner Lieblingsschwester Viola – sowie Ramona und Joseph, für ihre Geduld, ihre stetige Unterstützung und ihren Rückhalt. Besonders in den anspruchsvollen Phasen dieser Arbeit konnte ich mich stets auf euch verlassen. Für eure Geduld, euren Zuspruch und eure Unterstützung danke ich euch von Herzen.

Abschließend möchte ich all jenen danken, die mich auf meinem Weg begleitet, motiviert und unterstützt haben – auf fachlicher wie auf persönlicher Ebene.

Karlsruhe, im Mai 2025

Marco Stang

Inhaltsverzeichnis

1 Motivation und Ausgangslage

Im Zeitalter der Digitalisierung nimmt die Bedeutung von Künstliche Intelligenz (KI) stetig zu und signalisiert eine neue Entwicklung in unserer Gesellschaft. In allen Bereichen – von der Wissenschaft und Industrie bis zu staatlichen Einrichtungen – hat sich KI zu einem Kernthema entwickelt, das sowohl Begeisterung auslöst als auch essenzielle Fragen zur Gestaltung der Zukunft aufwirft. Einst als futuristisch betrachtet, ist KI bereits jetzt ein integraler Bestandteil unseres Alltags und wird in Zukunft zahlreiche Bereiche unseres täglichen Lebens beeinflussen. Im kommerziellen Sektor unterstützen KI-Systeme Kunden bei der Filmauswahl, prognostizieren den Energieverbrauch und steuern Smart-Home-Anwendungen wie Beleuchtung und Heizung entsprechend individueller Gewohnheiten. KI wird auch in anderen Bereichen Anwendung finden, die von der Entdeckung von neuen Planeten, der medizinischen Diagnose von Hautkrebs, der DNA-Analyse bis hin zur Erkennung von Freunden und Familie auf Fotos reichen.

Mit Blick auf die Energie- und Mobilitätswende werden Anstrengungen in der Entwicklung der künstlichen Intelligenz unternommen. Das hochautomatisierte Fahren ist neben der Elektrifizierung, der Mobilität als Dienstleistung (MaaS) (engl.: Mobility as a Service, MaaS) und der Vernetzung von Fahrzeugen oder Infrastruktur einer der Megatrends in der Automobilindustrie [1]. Für diese Anwendung werden eingebettete Systeme mit künstlicher Intelligenz eingesetzt, um Sensordaten von Kameras, Radar, Lidar, und weitere Sensoren zu kombinieren und der Fahraufgabe vollständig zu automatisieren[1]. Mit der fortschreitenden Entwicklung von Fahrerassistenzsystemen erhalten moderne Fahrzeuge, insbesondere jene mit Level 3-Autonomie, durch den Einsatz von KI Unterstützung in Funktionen wie der Erkennung von Verkehrszeichen und

[1] Nach Society of Automotive Engineers (SAE) wird dies als Autonomiestufe 5 bezeichnet. Die SAE stellt eine internationale Organisation dar, die die Autonomiestufen für die Automobilindustrie definiert.

Ampeln, dem Spurhalteassistenten sowie der adaptiven Geschwindigkeitsregelung. Neben teil- und hochautomatisierten Fahrzeugen wird die KI verwendet, um das Fahren komfortabler zu gestalten, sowohl für den Fahrer als auch für die Passagiere. Die Firma Tesla Inc. verwendet zum Beispiel anstelle eines konventionellen Regensensors zur Steuerung der Geschwindigkeit der Scheibenwischanlage ein künstliches neuronales Netz[2] (KNN), welches die Bilder der Frontkameras verwendet [2]. Das Infotainmentsystem Mercedes-Benz User Experience (MBUX) der Daimler AG stellt eine auf Komfortsysteme spezialisierte KI dar. Das System zeichnet Benutzerdaten in Bezug auf Mediennutzung, Fahrstrecken oder Radiolautstärke auf, generalisiert daraus individuelle Vorlieben und stellt dann die entsprechende Unterstützung dem Benutzer zur Verfügung [3].

Mit Blick auf die Investitionen auf europäischer Ebene plant die Europäische Kommission bis Ende 2020 20 Milliarden Euro und in den folgenden zehn Jahren weitere 20 Milliarden Euro jährlich für KI aufzubringen [4]. Der Investitionsplan der EU-Mitgliedstaaten, Schweiz und Norwegen sieht eine engere und effizientere Zusammenarbeit in vier Schlüsselbereichen vor: Steigerung der Investitionen, höhere Verfügbarkeit von Daten, Förderung von Talenten und Vertrauensbildung [4]. Eine engere Kooperation ist eine wesentliche Voraussetzung, wenn Europa bei der Entwicklung und dem Einsatz fortschrittlicher, ethischer und sicherer KI eine führende Position einnehmen will. Doch warum fasziniert KI sowohl Wissenschaft und Industrie und motiviert sogar ganze nationale und internationale Interessenverbände zur Kooperation und Bereitstellung von milliardenschweren Fördersummen? Andrew Ng, ehemaliger Baidu Chief Scientist, Coursera[3]-Mitbegründer und Stanford Professor äußerte sich dazu auf dem Stanford MSx Future Forum im Jahr 2017 [5] wie folgt:

> „Just as electricity transformed almost everything 100 years ago, today I actually have a hard time thinking of an industry that I don't think AI will transform in the next several years."

Ng zufolge ist KI eine der richtungweisenden Technologien, die, sobald sie entwickelt und verstanden wurde, eingesetzt werden kann, um verschiedene

[2] eine Untergruppe der KI

[3] Online-Lernplattform mit Fokus auf Machinelles Lernen (ML)

Wirtschaftszweige zu transformieren, vergleichbar wie die Entwicklung der Dampfmaschine oder die Entdeckung der Elektrizität. KI wird definiert als eine neue Generation von Technologien, die in der Lage sind, mit der Umwelt zu interagieren, indem sie (a) Informationen aus der Umwelt oder von anderen Computersystemen sammeln; (b) diese Informationen interpretieren und Muster erkennen, Regeln induzieren oder Ereignisse vorhersagen; (c) Ergebnisse erzeugen, Fragen beantworten oder anderen Systemen Anweisungen geben; und (d) die Ergebnisse ihrer Handlungen auswerten und ihre Entscheidungssysteme verbessern, um bestimmte Ziele zu erreichen [6]. In dem Maße, in dem die KI Fortschritte macht und Menschen und KI-Systeme zunehmend zusammenarbeiten, ist es unerlässlich, dass wir dem Ergebnis dieser Systeme vertrauen. Eine der meistzitierten Definitionen von Vertrauen ist:

> „the willingness of a party to be vulnerable to the actions of another party based on the expectation that the other will perform a particular action important to the trustor, irrespective of the ability to monitor or control that other party [7]".

Diese Definition beschränkt den Begriff des Vertrauens nicht nur auf die Mensch-Mensch-Interaktion und erlaubt es, Vertrauen im Hinblick auf die Technologie, einschließlich der KI, zu betrachten [8]. In Anbetracht der zunehmenden Integration von KI in sicherheitsrelevante Sektoren wie automatisiertes Fahren, Industrie 4.0 und das Gesundheitswesen, ergibt sich die Notwendigkeit, das Vertrauen in diese Systeme sowie in die von ihnen getroffenen Entscheidungen zu erhöhen. Die zentrale Fragestellung lautet daher, wie das Vertrauen der Menschen in algorithmische Entscheidungsprozesse gefördert werden kann. Zur Adressierung können verschiedene Strategien angewendet werden [9]:

- **Auditierung:** Externe Auditierungen validieren KI-Systeme hinsichtlich der Einhaltung spezifischer Auditrichtlinien und Entwicklungsstandards. Diese können in Übereinstimmung mit der Datenschutzgrundverordnung (DS-GVO), insbesondere Artikel 22 [10] („Automatisierte Entscheidungen im Einzelfall einschließlich Profiling"), entwickelt werden, der Bürgern ein Recht auf Erklärung algorithmischer Entscheidungen gewährt.
- **Erklärbarkeit:** Die Darlegung der Entscheidungsgrundlagen durch ein KI-System ermöglicht es Nutzern, Entscheidungen nachzuvollziehen,

eigenständig zu bewerten und basierend darauf Vertrauen zu entwickeln oder Entscheidungen abzulehnen.

- **schrittweise Einführung:** Die sukzessive Implementierung von Teilfunktionen bis zur Vollautomatisierung erlaubt es, anfangs unterstützende Aufgaben durch KI ausführen zu lassen und mit zunehmendem Vertrauen zu einer vollständigen Automatisierung überzugehen. Dieser Ansatz findet beispielsweise in der Entwicklung des automatisierten Fahrens gemäß den Stufen der SAE Anwendung.
- **Testen:** Testen stärkt das Vertrauen in KI-Systeme. Eine wesentliche Schwierigkeit beim Testen solcher Systeme liegt in der Identifizierung relevanter Testfälle aus potenzieller Szenarien sowie in der Erzeugung bisher unbekannter Testfälle, die im praktischen Einsatz auftreten könnten.

1.1 Klassische Testmethoden im Softwareentwicklungsprozess

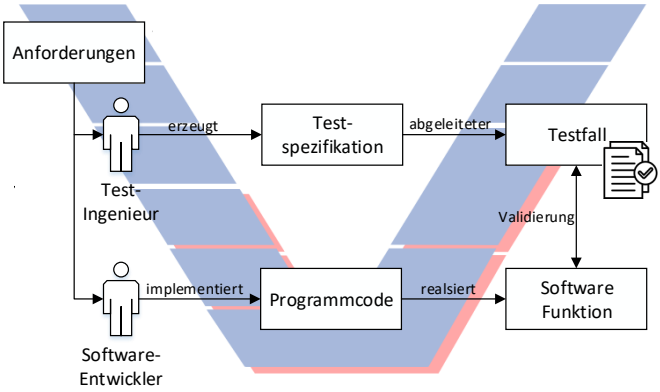

Abbildung 1.1: Schematische Darstellung des klassischen Testprozesses in der Softwareentwicklung

Klassische Testmethoden umfassen den systematischen Prozess der Überprüfung von Software, um deren Übereinstimmung mit den spezifizierten Anfor-

derungen sowie eine fehlerfreie Funktionalität zu gewährleisten. Diese Verfahren orientieren sich an sequenziellen Entwicklungsprozessen (siehe Abbildung 1.1), wobei die Testaktivitäten nach Abschluss der gesamten Softwareentwicklung einsetzen. Die initial definierten Anforderungen werden von Softwareentwicklern durch Programmcode in das gewünschte Verhalten überführt, wobei die auszuführenden Algorithmen gemäß der Programmierung die vorgesehenen Funktionen realisieren.

Definition 1.2 Software-Funktion: Der Begriff „Funktion“ wird in Anlehnung an das Konzept einer mathematischen Funktion verstanden. Demnach wird sie als eine Komponente der Software beschrieben, die Eingabedaten verarbeitet, eine spezifische Operation ausführt und ein Ergebnis ausgibt. [11]

Definition 1.4 Testfall: Eine Menge an Testeingaben und erwarteten Ergebnissen, die für ein bestimmtes Ziel entwickelt wurden, z.B. um einen bestimmten Programmpfad zu überprüfen oder um die Einhaltung einer bestimmten Anforderung zu verifizieren. [12]

Parallel zur Entwicklung der Software erstellt ein Testingenieur eine Testspezifikation, die für jede Anforderung einen zugehörigen Testfall vorsieht. Die Validierung prüft, ob das Entwicklungsergebnis den Anforderungen entspricht (siehe Gleichung 2.12). Ein Funktionsfehler deutet dabei auf eine inkorrekte Implementierung der Anforderungen im Programmcode hin.

1.2 Validierung von selbstlernenden Systemen mit individuellen Benutzeraktionen

KI, insbesondere in Form selbstlernender Systeme, weist charakteristische Merkmale auf, die konventionelle Testverfahren (siehe Abschnitt 1.1) vor wissenschaftliche Herausforderungen stellen.

Definition 1.6 Selbstlernende Systeme: Selbstlernende Systeme, auch als maschinelle Lernsysteme (siehe Kapitel 2) bezeichnet, sind computergestützte Systeme, die fähig sind, aus Daten zu lernen und sich ohne menschliche Eingriffe oder explizite Programmierung für die Ausführung spezifischer Aufgaben zu verbessern. [13]

Diese Herausforderungen ergeben sich aus den folgenden Merkmalen:

- **Datenbasiertes Lernen** (siehe Abschnitt 2.2): Datenbasiertes Lernen beschreibt den Prozess, durch die Funktionen aus der Analyse von Datenmengen lernen, Muster zu identifizieren, Entscheidungen zu treffen oder Prognosen zu erstellen. Diese Methode nutzt Ansätze des maschinellen Lernens, darunter überwachtes, unüberwachtes und verstärkendes Lernen, um Algorithmen zu trainieren.
- **Automatische Optimierung** (siehe Unterabschnitt 2.2.2): Der Prozess der Optimierung in selbstlernenden Systemen beinhaltet die gezielte Anpassung interner Parameter, um die Genauigkeit von Vorhersagen oder Entscheidungen zu erhöhen. Diese Anpassung erfolgt automatisch durch den Einsatz von Algorithmen, welche die Systemleistung anhand definierter Kriterien wie der Minimierung von Fehlern oder der Steigerung der Präzision evaluieren und verbessern.
- **Verallgemeinerungsfähigkeit** (siehe Unterunterabschnitt 2.2.2): Die Generalisierungsfähigkeit selbstlernender Systeme charakterisiert die Kompetenz, erworbene Kenntnisse und identifizierte Muster aus dem Trainingsdatensatz neuartige, unbekannte Daten zu übertragen. Diese Eigenschaft reflektiert die Leistungsfähigkeit eines Algorithmus hinsichtlich der Applikation seines erlernten Wissens auf vergleichbare, jedoch während der Trainingsphase nicht direkt erlebte Situationen.

Die Fortschritte im Bereich der selbstlernender Systeme eröffnen die Möglichkeit, Funktionen zu gestalten, die individuell auf die Bedürfnisse der Nutzer abgestimmt sind und somit über das Spektrum standardisierter Lösungen hinausgehen.

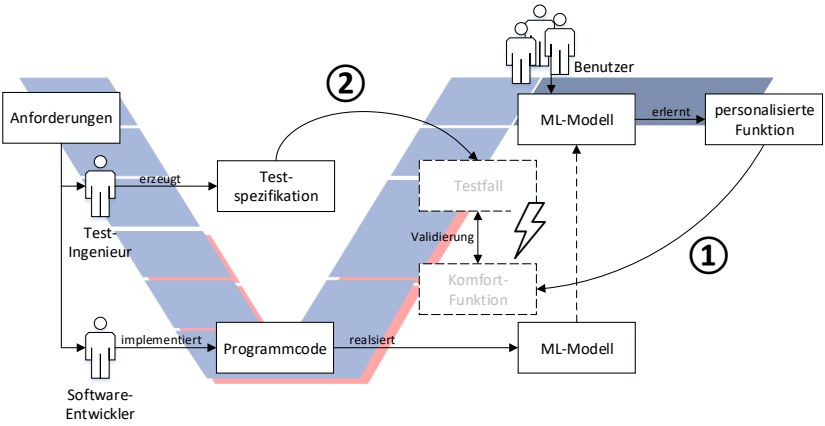

Abbildung 1.2: Darstellung des Testprozesses für Selbstlernende Systeme mit Benutzeraktionen und die daraus resultierenden Herausforderungen.(vergleiche Abbildung 1.1)

Definition 1.8 Selbstlernende Systeme mit individuellen Benutzeraktionen (SSIB): SSIB stellen eine spezifische Untergruppe der selbstlernenden Systeme dar, die sich darauf spezialisieren, aus den Aktionen einzelner Nutzer zu lernen. Anders als selbstlernende Systeme, die aus aggregierten Daten allgemeingültige Muster analysieren, konzentrieren sich SSIB auf die Verarbeitung und Analyse individueller Nutzeraktionen und die Extraktion von individuellen Muster.

SSIB analysieren Benutzerdaten wie Verhaltensmuster, Präferenzen und historische Aktivitäten, um individuelle Benutzerprofile zu erstellen. Basierend auf diesen Profilen werden die Funktionen des selbstlernenden Systems automatisch an die Bedürfnisse des einzelnen Nutzers angepasst, um personalisierte Erfahrungen wie maßgeschneiderte Empfehlungen, angepasste Benutzeroberflächen oder individuell relevante Inhalte zu bieten. Die Testmethodik für selbstlernende Systeme, die individuelle Benutzeraktionen integrieren, unterscheidet sich grundlegend von konventionellen Systemen (siehe Abbildung 1.2). Diese Unterschiede lassen sich wie folgt spezifizieren:

1. **Entstehung personalisierter Funktionen:** Die Entstehung der zu evaluierenden personalisierten Funktion erfolgt nach dem Abschluss des

Entwicklungsprozesses, induziert durch den adaptiven Lernprozess basierend auf dem Verhalten eines individuellen Nutzers. Dieser Prozess charakterisiert eine Abweichung von der klassischen Softwareentwicklung, bei denen die zu testenden Funktionen bereits während der Entwicklung implementiert wurde. Im Kontext SSIB entsteht die personalisierte Funktion als Ergebnis eines iterativen Lernvorgangs, der Daten und Interaktionen des Benutzers einbezieht. (Abbildung 1.2, dargestellt durch ①)

2. **Individualität der Testfälle:** Die Individualisierung der Testfälle ist eine direkte Konsequenz der personalisierten Funktion, die auf die Bedürfnisse einzelner Nutzer zugeschnitten ist. Um die Effektivität und Zuverlässigkeit dieser personalisierten Funktionen zu gewährleisten, ist die Erzeugung von individuellen Testfällen, die auf die Anforderungen und das Verhalten jedes Nutzers abgestimmt sind, unabdingbar. Die Entwicklung einer Methodik, die in der Lage ist, die Diversität individueller Benutzererfahrung adäquat zu adressieren, erfordert einen tiefgreifenden Einblick in die Dynamik zwischen Nutzerverhalten und Systemreaktion. (Abbildung 1.2, dargestellt durch ②)

3. **Dynamisches Verhalten der Testfälle:** Das Verhalten konventioneller Software ist in der Regel durch Anforderungsspezifikationen eindeutig definiert und verändert sich nur durch bewusste Aktualisierungen, die von den Entwicklern vorgenommen werden. Im Gegensatz dazu sind SSIB) durch ihre Fähigkeit zur stetigen Anpassung an neue Daten charakterisiert, was zu einem dynamischen Verhaltensmuster führt. Diese inhärente Dynamik erfordert Aufmerksamkeit bei der Implementierung selbstlernender Systeme, vor allem wenn diese darauf abzielen, sich an wandelndes Benutzerverhalten anzupassen. (Abbildung 1.2, dargestellt durch ②)

4. **Datenbasis als Testfokus**: Im Gegensatz zu traditionellen Softwaretests, die primär auf die Identifizierung und Korrektur von Programmcodefehlern fokussiert sind, erfordert die Evaluation von SSIB eine weitreichendere Teststrategie. Die Charakteristik selbstlernender Systeme, insbesondere die Erzeugung personalisierter Funktionen, resultiert nicht allein aus der algorithmischen Implementierung, sondern maßgeblich aus den Charakteristika der bereitgestellten Datensätze. Demzufolge ist eine fehlerfreie Implementierung eines ML-Modells nicht zwangsläufig

ein Indikator für dessen zufriedenstellende Funktionsfähigkeit. Vielmehr spielt die Qualität der zugrunde liegenden Datenbasis eine kritische Rolle für die Performance des Modells.

Definition 1.10 ML-Modell: Ein ML-Modell repräsentiert die spezifische Realisierung eines Lernalgorithmus innerhalb eines selbstlernenden Systems, welches darauf abzielt, durch die Analyse von Datenbeständen Muster zu identifizieren und basierend auf diesen Erkenntnissen Prognosen oder Entscheidungen zu generieren.

Gegenwärtig gestaltet sich die Erstellung eines Testorakels für SSIB als anspruchsvoll, da die gewünschten Eigenschaften formal nicht zu spezifizieren sind.

Definition 1.12 Testorakel: Ein Testorakel ist eine Informationsquelle, die das erwartete Ergebnis eines Testfalls definiert. Dieses erwartete Ergebnis wird mit dem tatsächlichen Ergebnis des Tests verglichen, um festzustellen, ob der Test bestanden oder nicht bestanden wurde [14].

Definition 1.14 Testorakelproblem: Das Testorakelproblem bezeichnet die Herausforderung bei der Validierung von Software, eine Methode oder Instanz zu finden, die eindeutig bestimmen kann, ob das Verhalten eines Systems bei einem gegebenen Testfall (nach Definition 1.4) korrekt ist oder nicht. [15]

Die Identifizierung eines Testorakels für Teilsysteme ist bereits zeitaufwendig und setzt domänenspezifisches Wissen voraus. Bei SSIB gestaltet sich die Erstellung geeigneter Testorakel als herausfordernd, da jeder Nutzer individuelles Verhalten zeigt. Daher können traditionelle Testmethoden nicht alle möglichen Benutzeraktionen abdecken. Zusätzlich erschwert die adaptive Natur selbstlernender Systeme, die ihr Verhalten während des Einsatzes an das des jeweiligen Nutzers anpassen, die Bestimmung eines Testorakels. Diese Dynamik in der Benutzeraktion und die kontinuierliche Anpassung der Systemfunktionen erfordern innovative Ansätze zur Testorakelerzeugung, die die spezifischen Eigenschaften von SSIB berücksichtigen und adressieren.

Die Herausforderung wird deutlich am Beispiel eines selbstlernenden Systems für Komfortfunktionen im Automobilbereich. Die selbstlernende Funktion hat unter anderem die Aufgabe, die Verwendung der Sitzheizung zu automatisieren. Jeder Fahrzeugkäufer nutzt die Sitzheizung auf unterschiedliche Weise, was zu personenspezifische Testfällen führt. Zudem verändert sich das Verhalten und somit die Spezifikation der Testfälle mit den Jahreszeiten, Wetterbedingungen und dem subjektiven Empfinden des Fahrers. Beispielsweise korreliert das Benutzerverhalten im Sommer nicht mit dem im Winter.

1.3 Wissenschaftliche Fragestellung der Dissertation und abgeleitete Forschungsfragen

Der zentrale Forschungsbereich dieser Dissertation konzentriert sich auf die Validierung von selbstlernenden Systemen mit individuellen Benutzeraktionen (siehe Abbildung 1.3). Der Forschungsbereich wird in drei spezifische Forschungsfragen unterteilt, welche die Basis für eine strukturierte Auseinandersetzung mit den verschiedenen Aspekten der Validierung selbstlernender Systeme bilden:

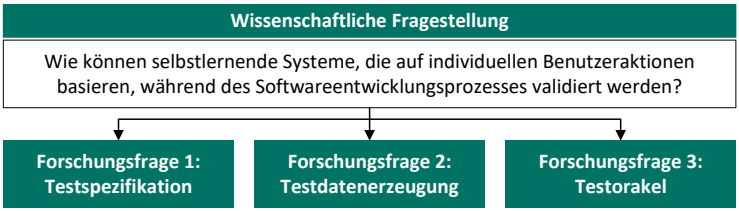

Abbildung 1.3: Wissenschaftliche Fragestellung der Dissertation und daraus resultierende Forschungsfragen

Forschungsfrage 1: Testspezifikation

Wie können wiederverwendbare Testfälle für personalisierte Systeme mit Benutzeraktionen spezifiziert werden?

Forschungsfrage 2: Testdatenerzeugung

Wie können synthetisch realistische Testdaten mit personenspezifischen Benutzeraktionen erzeugt werden?

Forschungsfrage 3: Testorakel

Wie kann die Ambiguität im individuellen Benutzerverhalten selbstlernender Systeme durch ein Testorakel abgebildet werden?

1.4 Struktur und Anforderungen an die Validierung von selbstlernenden Systemen mit individuellen Benutzeraktionen

Der Systementwurf umfasst drei Hauptbereiche, Szenariodefinition, Szenarioerzeugung und Validierung, (siehe Abbildung 1.4) zu jedem dieser Bereiche werden im folgenden Anforderungen beschrieben. Eine Übersicht über die Abdeckung der Anforderungen mit einer Kurzbeschreibung des Lösungsansatzes und einem Verweis auf die entsprechenden Abschnitte findet sich in den Tabellen 12.7 bis 12.8 auf den Seiten 224-225.

Szenariodefinition

Der Einsatz von Szenarien zum Testen ist eine etablierte Strategie, um die Zuverlässigkeit und Funktionalität von Software- und Systementwicklung sicherzustellen (siehe Kapitel 6). Dabei dienen Szenarien als Ausgangspunkt für die Ableitung von Testfällen. Bisherige Ansätze konzentrieren sich auf Funktionen im Kontext des automatisierten Fahrens, bei denen keine Personalisierung zwischen Benutzer und Funktion erforderlich ist. In der ersten Forschungsfrage wird betrachtet, welche zusätzlichen Ebenen in die bestehende Szenariodefinition integriert werden müssen, um eine Funktion zu beschreiben, die sich individuell auf einen Benutzer anpasst. Dabei werden geeignete Methoden und Techniken zur Erfassung und Integration von Benutzeraktionen in die Szenariobeschreibung untersucht, um wiederverwendbare Testfälle zu erstellen, die die individuelle Anpassungsfähigkeit der Funktion berücksichtigt. Basierend

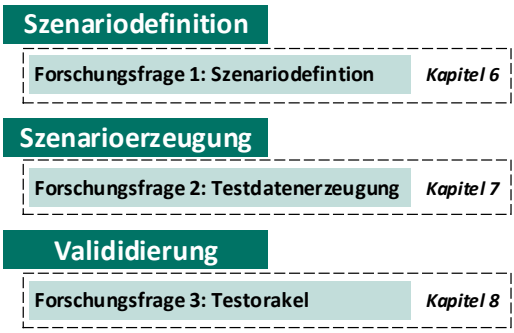

Abbildung 1.4: Der Systementwurf untergliedert sich in drei Bereiche: Szenariodefinition, Szenarioerzeugung und Metamorphes Testen, welche den Forschungsfragen 1 bis 3 entsprechen und jeweils die Testspezifikation, Testdatenerzeugung und Testorakel behandeln.

auf der Forschungsfrage 1 - Testspezifikation ergeben sich folgende Anforderungen an die Szenariodefinition.

Anforderungen an die Szenariodefinition

- **Anforderung 1** (AS1): Aufbauend auf der Szenariodefinition nach ISO 34501, muss die Szenariodefinition für die Beschreibung eines Szenarios für selbstlernende Systeme mit Benutzeraktion angepasst werden.
- **Anforderung 2** (AS2): Die Szenariodefinition soll die Identifikation von repräsentativen Szenarien ermöglichen, die die charakteristische Nutzung eines selbstlernenden Systems durch einen Benutzer abbilden.
- **Anforderung 3** (AS3): Die Szenariodefinition muss die Einbeziehung relevanter Randbedingungen und Variablen ermöglichen, welche das Verhalten der selbstlernenden Systemen beeinflussen könnte.

Szenarioerzeugung

Die Bereitstellung geeigneter Testdaten, die auf den definierten Szenarien basieren, ist entscheidend für die Überprüfung der Funktionalität von Systemen. Allerdings hat die Verwendung realer Testdaten Nachteile, da reale Szenarien

entweder nicht verfügbar sind oder ihre Anzahl begrenzt ist. Insbesondere für neuartige oder unerwartete Situationen fehlt eine ausreichende Datengrundlage, um die Funktionalität zu testen. Ferner gestaltet sich die Replikation seltener oder gefährlicher Situationen als aufwendig, da sie mit hohen Kosten, Risiken oder ethischen Bedenken verbunden sind. Ein weiterer Aspekt ist der Schutz sensibler Informationen. Reale Testdaten können personenbezogene Daten oder andere vertrauliche Informationen enthalten, die nicht ohne Weiteres für Testzwecke verwendet werden dürfen. Um den Mangel an realen Szenarien zu kompensieren, wird in dieser Dissertation untersucht, wie synthetische Testdaten generiert werden können, die realistische Szenarien abbilden, jedoch keine personenbezogenen Daten oder vertraulichen Informationen enthalten. Dabei werden Methoden und Techniken zur Erzeugung von Kontextdaten (siehe Abschnitt 7.1) und Benutzeraktionen (siehe Abschnitt 7.3) erforscht, um Testdaten zu schaffen, die die individuelle Anpassungsfähigkeit der Funktionen berücksichtigen.

Anforderungen an die Szenarioerzeugung

- **Anforderung 1** (AD1): Die Szenarioerzeugung muss einerseits den untersuchten Kontext darstellen und andererseits Benutzeraktionen abbilden.
- **Anforderung 2** (AD2): Die Erzeugung von Testdaten soll mithilfe eines dezentralen Ansatzes erfolgen, um ein modulares Hinzufügen oder Entfernen von Datenquellen zu ermöglichen.
- **Anforderung 3** (AD3): Die synthetisch erzeugten Testdaten sollen der empirischen Datenverteilung entsprechen, die in der realen Welt beobachtet wird.
- **Anforderung 4** (AD4): Die Szenarioerzeugung soll die Erzeugung von Testdaten ermöglichen, die ein realitätsnahes Verhalten eines Benutzers abbilden.

Validierung von Benutzeraktionen

Die Eigenschaft selbstlernender Systeme besteht darin, dass sie ihre Funktionalität basierend auf den ihnen zur Verfügung gestellten Datensätzen anpassen können. Dadurch kann sich das Verhalten einer selbstlernenden System indi-

viduell auf den jeweiligen Benutzer anpassen. Diese Anpassung erfolgt nicht während der Entwicklung der Funktion, sondern während ihrer Verwendung durch den Benutzer selbst. Jedoch stellt das Testorakelproblem, das sich aus der Ambiguität im individuellen Benutzerverhalten selbstlernender Systeme ergibt, eine Herausforderung für die Validierung dar. Einerseits kann ein Benutzer sein Verhalten verändern, und andererseits können verschiedene Benutzer im gleichen Szenario unterschiedliche Verhaltensweisen aufweisen (siehe Abbildung 5.1). Dadurch lässt sich kein eindeutiges und vordefiniertes korrektes Verhalten für die Funktion a priori[4] festlegen und entsprechend nicht verifizieren. Es ist erforderlich, eine Testmethode zu finden, die die Anforderungen an ein System zur Erfassung von Benutzeraktionen erfüllt und die Validierung ermöglicht (siehe Kapitel 9).

Anforderungen an die Validierung von Benutzeraktionen

Im vorliegenden Abschnitt werden die konzeptionellen Anforderungen für die Validierung eines selbstlernenden Systems mit Benutzeraktionen dargestellt:

- **Anforderung 1** (AM1): Das Validierungskonzept muss das Testorakelproblem verhindern, das insbesondere beim Testen selbstlernender Systeme mit Benutzeraktionen auftritt.
- **Anforderung 2** (AM2): Es müssen spezifische Eigenschaften identifiziert werden, die sich auf die Benutzeraktionen und das Verhalten der Funktion auswirken. Diese Eigenschaften sollten geeignet sein, um potenzielle Veränderungen im Systemverhalten aufzudecken.
- **Anforderung 3** (AM3): Die Testergebnisse sollen so gestaltet sein, dass sie eine automatisierte Analyse und Interpretation ermöglichen.

[4] Im Gegensatz zu spezifischen Testorakel aus Abschnitt 3.1

2 Selbstlernende Systeme und Testmethoden

Selbstlernende Systeme (nach Definition 1.6) lassen sich in die Kategorien der starken (englisch: „strong AI“ oder „general AI“) und der schwachen (englisch: „weak AI“ oder „narrow AI“) KI einordnen. Eine starke KI zeichnet sich durch ihre Fähigkeit aus, in sämtlichen Bereichen menschlicher kognitiver Leistungen gleichwertige oder eine überlegene Performanz zu erbringen. Dies schließt Kapazitäten wie logisches Denken, Selbstbewusstsein, emotionale Reaktionen und die Fähigkeit zum domänenübergreifenden Problemlösen ein.

Aktuelle KI-Systeme erreichen diese Stufe der Intelligenz jedoch nicht. Gemäß einer Erhebung der Association for the Advancement of Artificial Intelligence, an der 80 führende KI-Forschende teilnahmen, vertreten 92,5% der Befragten die Ansicht, dass die Realisierung einer starken KI noch mindestens 25 Jahre entfernt ist und somit jenseits des gegenwärtig prognostizierbaren Zeitrahmens liegt. Im Gegensatz dazu repräsentieren die momentan verfügbaren KI-Systeme ausschließlich Ausprägungen einer schwachen KI. Unter schwacher KI wird ein System verstanden, welches die spezifisch für die Bearbeitung und Lösung festgelegter Aufgabenstellungen konzipiert ist. Diese Lösungsansätze basieren auf mathematischen und statistischen Methoden. Die resultierenden Systeme sind dazu befähigt, sich durch die Analyse von Trainingsdaten eigenständig zu verbessern, erreichen dabei allerdings lediglich eine oberflächliche Form der Intelligenz, die kein tiefergehendes Verständnis der Problemlösung impliziert. [16]

Zusammen mit dem Begriff der KI werden die Begriffe ML, Repräsentationslernen und Deep Learning (DL) genannt (siehe Abbildung 2.1). Die Kerntechnologie der künstlichen Intelligenz stellt das Forschungsgebiet des ML dar. ML hat das Ziel, Regeln aus Trainingsdaten abzuleiten und diese als Modelle darzustellen. Seit dem Aufkommen der ersten Computer werden mathematische Modelle zunehmend eingesetzt, um Menschen bei Entscheidungsprozessen zu

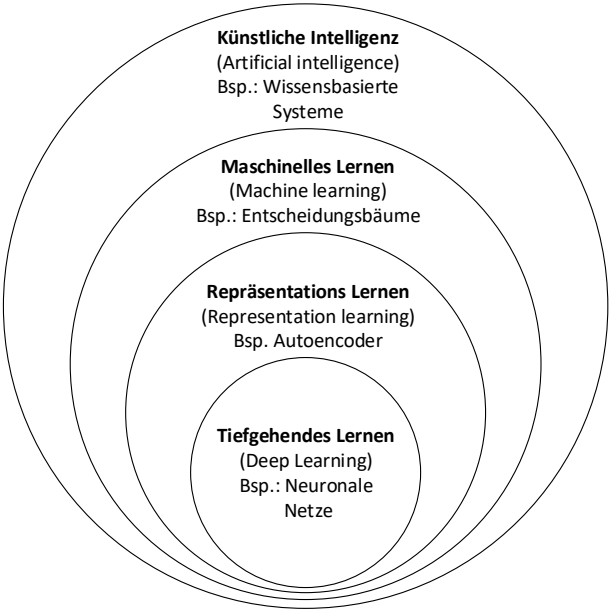

Abbildung 2.1: Venn-Diagramm zur Visualisierung der Untermengen der Künstlichen Intelligenz: maschinelles Lernen, Repräsentationslernen und Deep Learning (DL). Jeder Abschnitt des Venn-Diagramms enthält ein Beispiel für eine KI-Technologie (nach [17])

unterstützen. Bei der klassischen KI (Wissensbasierte Systeme) werden Regeln manuell vorgegeben und somit das Verhalten des Modells durch einen Experten direkt beschrieben. Methoden des maschinellen Lernens erzeugen eigenständig Regeln, basierend auf Trainingsdaten. Da dieser Vorgang von statistischer Natur ist, müssen hohe Ansprüche an die Datengüte gestellt werden, um korrekt funktionierende Modelle zu erhalten. Diese trainierten Modelle können danach auf neue, unbekannte Daten reagieren, sie in Kategorien klassifizieren, Vorhersagen zur Entwicklung treffen oder Handlungsvorschläge generieren [18]. Die für das Training von ML-Systemen verwendeten Daten stellen reale Beobachtungen der Umwelt dar. Nicht alle Beobachtungen tragen im gleichen Maße zu einem korrekten Ergebnis bei. Für die Vorhersage des Restwertes eines Fahr-

zeugs sind beispielsweise Beobachtungen über Kilometerstand, Unfallschäden oder Anzahl der Vorbesitzer relevanter als der Name des Vorbesitzers. Diese Beobachtungen werden Merkmale (oder engl. Features) genannt. Im Gegensatz zum maschinellen Lernen, das eine manuelle Vorgabe von Merkmalen erfordert, ist das Repräsentationslernen durch eine automatische Darstellung von Merkmalen charakterisiert. Die erlernten Darstellungen von Merkmalen resultieren in einer höheren Performanz, im Vergleich zu manuell erarbeiteten Darstellungen. DL ist ein Teilbereich des Repräsentationslernens, der sich mit Algorithmen beschäftigt, die von der Struktur und Funktion des Gehirns inspiriert sind und Künstliche Neuronale Netze (KNN) genannt werden.

Obwohl die Grundlagen des DL bereits in den 1980er Jahren formuliert wurden, hat diese Technologie erst in jüngerer Vergangenheit eine breite praktische Anwendung gefunden. Dies ist vor allem auf zwei wesentliche Entwicklungen zurückzuführen: Erstens die Verfügbarkeit umfangreicher Datenmengen und zweitens die Bereitstellung leistungsfähiger Hardware, insbesondere Grafikprozessoren. DL zeichnet sich durch seine Kapazität aus, verborgene Strukturen innerhalb von Datensätzen zu identifizieren, indem es automatisierte, abstrakte Repräsentationen der Daten über mehrere Verarbeitungsebenen hinweg generiert. Die initiale Ebene in einem solchen Netzwerk, bekannt als die sichtbare Schicht, bearbeitet direkt beobachtbare Daten, wie Pixelwerte in Bildern oder Sensorikmesswerte. Auf diese sichtbare Schicht folgen mehrere versteckte Schichten, die zunehmend abstraktere Merkmale aus den Daten extrahieren. Insbesondere Convolutional Neural Network (CNN) haben Fortschritte im Bereich des maschinellen Sehens ermöglicht und gelten als ein Schlüsselbeitrag zur modernen KI-Forschung [17].

2.1 Algorithmen für selbstlernende Systeme

Die Hauptaufgabe eines ML-Algorithmus liegt darin, aus Trainingsdaten Muster abzuleiten. Das maschinelle Lernen impliziert den Lernvorgang, bei dem Muster in Datensätzen gefunden werden, die für die Bewertung oder Vorhersage eines bestimmten Ereignisses relevant sind [19]. ML-Algorithmen erweisen sich als vorteilhaft für die Entwicklung von Systemen, die sich aufgrund der Unvollständigkeit ihrer Informationen nicht durch regelbasierte Ansätze beschreiben und somit nicht mittels traditioneller Programmierung realisieren lassen [20].

Traditionell differenziert die Forschung im Bereich des maschinellen Lernens (ML) zwischen drei Hauptkategorien von Lernalgorithmen: überwachtes, unüberwachtes und semi-überwachtes Lernen [21]. Diese Klassifizierung basiert auf der Methodik, nach der der Algorithmus trainiert wird, um Muster innerhalb der Traininsdaten zu identifizieren.

Überwachtes Lernen bezieht sich auf das Entwickeln eines statistischen Modells für Probleme, bei denen sowohl die Eingabemerkmale als auch die dazugehörigen Zielvariablen, die sogenannten Labels, bekannt sind. Der Trainingsdatensatz dient dazu, die Parameter des Modells so anzupassen, dass es die Zielvariable vorhersagen kann. Nach der Trainingsphase wird ein Validierungsdatensatz genutzt, um die Präzision des Modells zu evaluieren [22].

Die Effektivität des Modells wird durch den Vergleich der Vorhersagen des Algorithmus mit den Bewertungen von Experten beurteilt. Überwachte ML-Algorithmen finden Anwendung in der Klassifikation, um die Zugehörigkeit zu spezifischen Klassen vorherzusagen, oder in der Regression, um kontinuierliche Werte, wie beispielsweise Aktienkurse, zu prognostizieren.

Die Klassifizierungsaufgabe von überwachten ML-Algorithmen lässt sich nach [23] in vier Untergruppen unterteilen. Dabei sind Datenpunkte $x1; ...; x_n$ in vordefinierte Klassen einzuordnen $C1; ...; C_l$.

- **Binäre Klassifikation:** die Eingangsdaten werden in exakt eine von zwei sich nicht überschneidenden Klassen $(C1; C2)$ eingeordnet. Die binäre Klassifizierung ist die beliebteste Klassifizierungsaufgabe. Die zugewiesenen Kategorien können objektiv und unabhängig von einer manuellen Bewertung sein (wie z. B. bei Wahlergebnissen) oder subjektiv und abhängig von einer manuellen Bewertung (z. B. positive oder negative Bewertungen bei Amazon.com). Wenn das überwachte Lernen z. B. zum Trainieren eines Spamfilters verwendet wird, versucht der Algorithmus, eine Funktion zu erlernen, die die Attribute, die eine E-Mail beschreiben, auf einen Wert (Spam/Nicht-Spam) für das Zielattribut abbildet.
- **Mehrklassen Klassifikation:** die Eingabedaten sollen in exakt eine, von l sich nicht überlappenden Klassen klassifiziert werden. Eines der bekanntesten Mehrklassenproblemen ist die Identifizierung des Iris-Typs

in einem Drei-Klassen-Datensatz[1]. [24]. Wie im binären Fall kann die Multiklassen-Kategorisierung objektiv oder subjektiv sein.

- **Multilabel Klassifikation:** die Eingabedaten werden in mehrere von l nicht überlappenden Klassen C_j klassifiziert. Beispiele hierfür ist die Identifizierung von Szenen aus Bilddaten. In [25] wird ein aus sechs Klassen bestehender Datensatz betrachtet. Jede Klasse entspricht einer von sechs verschiedenen Szenen, z. B. Strand, Sonnenuntergang usw. Diese Szenen können sich in einem Bild überschneiden, d. h. ein Bild kann gleichzeitig zu mehreren Klassen zugehörig sein.
- **Hierarchische Klassifikation**: Die Eingabe ist in exakt eine Klasse C_j zu klassifizieren, die in Unterklassen unterteilt oder in Überklassen gruppiert werden kann. Die Hierarchie ist festgelegt und kann während der Klassifizierung nicht geändert werden.

Unüberwachte Lernalgorithmen extrahieren Muster aus einem Datensatz ohne die Verwendung von Label. Im Unterschied zu überwachtem Lernen, das direkt auf Regressions- oder Klassifikationsprobleme anwendbar ist, da die korrekten Ausgabewerte bekannt sind, fokussiert sich das unüberwachtes Lernen auf die Entdeckung der inhärenten Struktur der Daten. In diesem Zusammenhang werden drei Typen von Datenrepräsentationen (und zugehörigen Algorithmen) differenziert, um die zugrunde liegenden Muster in den Daten zu identifizieren.

- **Clusteranalyse** (auch: Agglomerationsanalyse) bezeichnet die Gruppierung von Datenpunkten auf Basis von Ähnlichkeitsmetriken zu einem gemeinsamen, vereinheitlichten Cluster [26].
- **Dimensionsreduktion** ist eine klassische Aufgabe der Datenvorverarbeitung und hat zum Ziel, die Dimension der Daten zu reduzieren und gleichzeitig relevante Informationen beizubehalten [26].
- **unüberwachte Anomaliedetektion** kann selbstständig Datenpunkte erkennen, deren Eigenschaften sich von der Norm der anderen Datenpunkte abhebt und dadurch einen Ausreißer (engl. outlier) darstellt [27].

[1] Der Datensatz besteht aus jeweils 50 Beispielen von drei Irisarten (Iris setosa, Iris virginica und Iris versicolor)

Semi-überwachtes Lernen ist ein Ansatz im maschinellen Lernen, der Techniken des überwachten und des unüberwachten Lernens kombiniert, um die Genauigkeit des Lernprozesses zu verbessern. Bei diesem Ansatz wird eine Mischung aus annotierten (d.h. mit Labels versehenen) und nicht-annotierten Daten verwendet. Die Grundidee besteht darin, dass das ML-Modell durch die Verwendung der annotierten Daten grundlegende Muster und Zusammenhänge erlernt, während die nicht-annotierten Daten dazu beitragen, diese Muster zu verfeinern und zu generalisieren, um eine bessere Gesamtleistung des Modells zu erzielen.

2.2 Modelle für selbstlernende Systeme

Ein maschineller Lernalgorithmus erstellt durch das Lernen aus Daten ein ML-Modell (nach Definition 1.10), das die erkannten Muster und Beziehungen zur Vorhersage neuer Datensätze nutzt. Dieses ML-Modell repräsentiert die erlernten Zusammenhänge als Funktion, die Eingabeattribute auf Zielattribute abbildet. Ein ML-Modell kann als mathematische Gleichung betrachtet werden, die die Beziehung zwischen einer oder mehreren Eingangsvariablen X und einer Ausgangsvariable Y definiert. Diese Beziehung bildet die Grundlage für die Generierung von Vorhersagen oder Entscheidungen auf Basis neuer, bisher unbekannter Daten.

$$Y = f(X) \tag{2.1}$$

Durch den Trainingsprozess erlernt der ML-Algorithmus die spezifische Zielabbildungsfunktion f, die den Zusammenhang zwischen Eingangs- und Ausgangsvariablen festlegt. Die Struktur der Abbildungsfunktion ist a priori nicht bekannt, sodass verschiedene ML-Algorithmen angewandt und evaluiert werden, um festzustellen, welcher Algorithmus die zugrunde liegende Funktion bestmöglich approximieren kann. [28]

Zusätzlich zu der klassischen Unterscheidung von ML-Algorithmen in überwacht und unüberwacht kann ebenfalls eine Differenzierung in instanzbasierte und modellbasierte ML-Algorithmen vorgenommen werden (Abbildung 2.2). Diese Klassifikation unterscheidet nicht zwischen dem Vorhandensein von

Labeln und somit dem Lernverfahren, sondern über die Eigenschaften des resultierenden ML-Modells und seiner Abbildungsfunktion. [29]

Algorithmen, die keine Annahmen über die Anzahl der Parameter der Abbildungsfunktion machen, werden instanzbasierte Algorithmen (engl. non parametric machine learning algorithms) genannt. Die Anzahl der Parameter wird erst zur Laufzeit des Trainings festgelegt und kann sich mit jeder neuen Trainingsinstanz verändern. Im Gegensatz zu den modellbasierten ML-Algorithmen wird kein Modell mit explizit extrahiertem Wissen erzeugt, sondern dieses wird aus abgespeicherten Trainingsinstanzen generiert. Instanzbasierte ML-Algorithmen vergleichen neue Probleminstanzen mit gespeicherten Trainingsinstanzen bewertet die Ähnlichkeit dieser, anhand einer Distanzmetrik. [28]

Bei modellbasierten ML-Algorithmen wird eine feste Anzahl an Parameter für eine Abbildungsfunktion (z.B. $f(X) = a * x^3 + b * x^2 + b * x + c$) über ein Lernverfahren anhand von Trainingsdaten erlernt. Die Anzahl der Parameter der Abbildungsfunktion ist unabhängig von der Anzahl der Trainingsbeispiele. Aus diesem Grund werden modellbasierte ML-Algorithmen als parametrische ML-Algorithmen (engl. parametric machine learning algorithms) bezeichnet. [28]

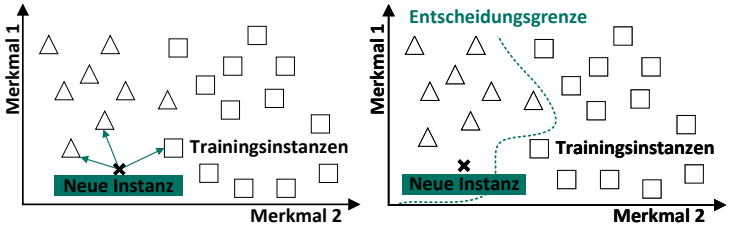

Abbildung 2.2: Darstellung von instanzbasierten ML-Algorithmen (links) und modelbasierten ML-Algorithmen (rechts)

Die verbreitetsten Vertreter für parametrische Algorithmen des maschinellen Lernens sind: Logistische Regression, Lineare Diskriminanzanalyse, Naive Bays und künstliche neuronale Netze. Besonders die zuletzt genannten künstlichen neuronalen Netze haben zum aktuellen Erfolg des maschinellen Lernens beigetragen und werden in Unterabschnitt 2.2.1 betrachtet. Diese Grundlagen dienen für die in Kapitel Abschnitt 8.4 entwickelten ML-Modelle.

2.2.1 Künstliche Neuronale Netze

Neuronale Netze in der Biologie bestehen aus vernetzten Nervenzellen, die das Nervensystem formen. Das menschliche Gehirn, mit seinen 10 bis 100 Milliarden Neuronen, ist auf die effiziente Verarbeitung sensorischer Reize spezialisiert [17]. Durch die Fähigkeit des Gehirns zu lernen und zu bewerten, sind Organismen in der Lage, sich an wechselnde Umweltbedingungen anzupassen und in neuen Situationen zu funktionieren.

KNNs zielen darauf ab, die Funktionen biologischer Nervensysteme nachzubilden, indem sie aus Basiseinheiten zusammengesetzt werden, die den biologischen Neuronen entsprechen. Diese künstlichen Neuronen sind in der Lage, mehrere Eingangssignale in ein Ausgangssignal zu überführen. Obwohl einzelne Neuronen lediglich über eine begrenzte Rechen- und Klassifikationskapazität verfügen, ermöglicht erst ihre strukturierte Anordnung in Schichten die Leistungsfähigkeit.

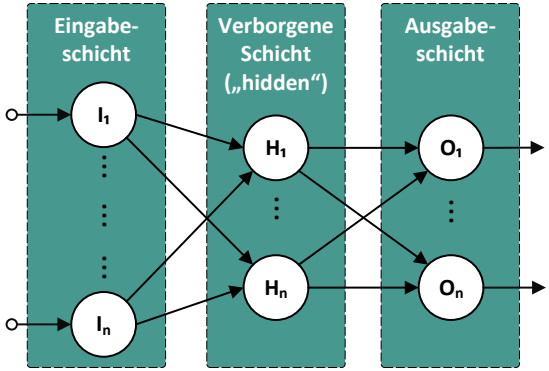

Abbildung 2.3: Verkettung von einzelnen künstlichen Neuronen zu einem Multilayer Perceptron (MLP) bestehend aus Eingabeschicht, verborgene Schicht (hidden Layer) und Ausgabeschicht.

Die ersten Schritte zur Entwicklung und Anwendung eines künstlichen Modells menschlicher Neuronen wurden zwischen 1943 und 1958 vorgenommen [30], [31]. 1958 stellte Rosenblatt das Perzeptron vor, einen binären Klassifikator, der einen Eingangsvektor $\vec{x}$ auf zwei Klassen, 1 und 0, abbildet:

$$f(\vec{x}) = \begin{cases} 1, & \text{falls } \vec{w}^T \vec{x} + b \geq 0 \\ 0, & \text{sonst.} \end{cases} \tag{2.2}$$

Abhängig von den Gewichten $\vec{w}$ und dem Bias b (siehe Defintion 2.4) kann das Perzeptron für einen gegebenen Eingangsvektor $\vec{x}$ entweder „feuern“ (1 als Ausgang produzieren) oder „nicht feuern“ (0 als Ausgang produzieren) (siehe Abbildung 2.4).

Dieses Konzept lehnt sich an das Verhalten biologischer Neuronen an, welche ein Aktionspotenzial nur dann auslösen, wenn die Summe der eingehenden Signale einen spezifischen Schwellenwert überschreitet. Aufgrund seiner linearen Natur kann dieses Modell allerdings die Ausgänge realer, in der Regel nicht linearer Prozesse nicht zuverlässig prognostizieren. In den frühen 1960er Jahren wurden zwei bedeutende Fortschritte gemacht, um diese Limitation zu überwinden:

Im Jahr 1961 adaptierte Steinbuch das Konzept des künstlichen Neurons weiter und entwickelte die Lernmatrix [32]. Diese bestand aus einer Matrix aus bipolar angeordneten Relais und Signaldrähten, die es ermöglichte, bedingte Reflexe zwischen den Signalleitungen mithilfe der Prinzipien künstlicher Neuronen zu schaffen.

Im Jahr 1962 stellte Rosenblatt eine Lösung für nicht lineare Problemstellungen vor, indem er Perceptrons schichtweise anordnete und mehrere dieser Schichten miteinander verband, was zur Entdeckung des MLP) [33] führte (siehe Abbildung 2.3).

Alle Neuronen einer Schicht propagieren ihre Ausgabe an jedes Neuron der folgenden Schicht über eine gewichtete Verbindung. Daraus ergibt sich die Gewichtsmatrix w, die eine Schicht mit n Neuronen mit einer Schicht mit m Neuronen verbindet und somit eine Größe von $n \times m$ hat. Eine einzelne Schicht des MLP kann als lineare Transformation mit folgender Transformation bezeichnet werden.

$$f(\vec{x}) = w\vec{x} + \vec{b} \tag{2.3}$$

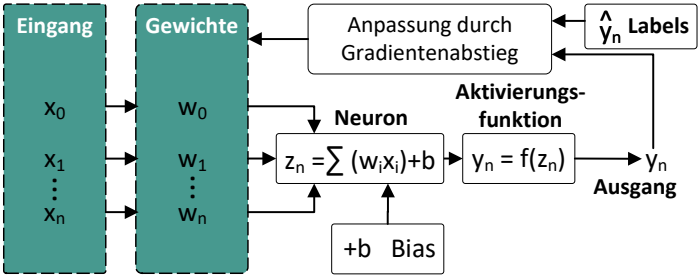

Abbildung 2.4: Funktionsweise eines künstlichen Neurons mit der Summierung der gewichteten Eingänge und der Aktivierungsfunktion zur Berechnung der Ausgaben.

Die Funktionsweise eines künstlichen Neurons ist in Abbildung 2.4 dargestellt. Ein künstliches Neuron besitzt eine definierte Anzahl an Eingaben x (Input) und eine Anzahl an Ausgaben y (Output). Die Funktion eines Neurons ist es, aus dem Input einen Output zu berechnen. Dazu wird zuerst eine Summe $\sum w_i x_i + b = z$ gebildet, wobei w_i gleich dem Gewicht der Eingabe x_i entspricht und b dem Offset. Zusätzlich wird dieser Summe eine Aktivierungsfunktion $f(z) = y$ nachgeschaltet, die die zuvor summierten und gewichteten Eingaben mit einer nicht linearen Funktion verändert und als Ausgang zur Verfügung stellt. Durch die Auswahl der Aktivierungsfunktion können verschiedene Verfahren optimiert oder die Funktionsapproximation überhaupt erst ermöglicht werden. Der Ausgang y wird im Falle des überwachten Lernens mit den Labels $\hat{y}$ verglichen. Ein Gradientenabstiegsverfahren adaptiert basierend auf diesem Fehler iterativ die Gewichte und den Bias des künstlichen neuronalen Netzes an. Dieser Vorgang wird als Training bezeichnet, das künstliche neuronale Netz „lernt'".

Definition 2.2 Gewichte: In KNN sind Gewichte veränderbare Parameter, die Signale durch Multiplikation verstärken oder abschwächen. Sie bilden die Basis für die gewichteten Verbindungen zwischen Neuronen (siehe Abbildung 2.4) und ermöglichen zusammen mit Biases die Anpassung durch den Lernalogrithmus. [34]

Definition 2.4 Bias: Der Bias ist ein skalarer Wert, welcher auf den Signalwert x addiert wird (siehe Gleichung 2.3). Durch den Bias lässt sich eine Art Schwellwert für ein Signal erstellen, welcher beispielsweise bei einer ReLU Aktivierungsfunktion verwendet werden kann. Mit einem Bias können die Aktivierungsfunktionen somit angepasst und auf das gegebene Zielsystem hin optimiert werden. [34]

Da die Operationen in einem KNN grundsätzlich linear sind, resultieren lineare Kombinationen dieser Operationen in linearen Transformationen, was ihre Effektivität für nicht lineare Probleme einschränkt. Um die Fähigkeit des Netzes zur Vorhersage nicht linearer Prozesse zu verbessern, ist die Einführung zusätzlicher Nichtlinearitäten erforderlich.

Zu diesem Zweck werden sogenannte Aktivierungsfunktionen φ auf die Ausgänge aller Schichten des Netzes angewendet, woraus der gefilterte Schichtausgang $\varphi f(x)$ hervorgeht. In einem MLP, das aus mehreren verketteten Schichten besteht, wobei jede Schicht die Ausgabe der vorherigen verarbeitet, lässt sich die Funktionsweise des gesamten Netzes mit n Schichten durch folgende Gleichung darstellen:

$$y(z_0) = \varphi_n(f_n(...\varphi_2(f_2(\varphi_1(f_1(z_0))))) \tag{2.4}$$

Entsprechend der Abbildung 2.4 stellt φ_n die n-te Aktivierungsfunktion und f_n der Ausgang des n-ten Neurons dar.

2.2.2 Aufbau eines Lernalgorithmus

Ein zentrales Anliegen beim Einsatz maschineller Lernsysteme besteht darin, eine Menge von Parametern Θ zu identifizieren, die es ermöglichen, das zugrundeliegende Lernproblem effektiv zu generalisieren.

Goodfellow et al. [17] definieren ein Lernproblem durch die Erfahrung E, repräsentiert durch eine Sammlung von Trainingsdaten bestehend aus n Beobachtungen $\mathbb{X} = \{x^{(1)}, ..., x^{(n)}\}$ und, unter der Prämisse eines überwachten Lernansatzes, zugehörigen Ground-Truth-Labels $\mathbb{Y} = \{y^{(1)}, ..., y^{(n)}\}$. Das primäre Ziel eines Lernalgorithmus ist es, basierend auf diesem Datensatz, ein Modell $f_{Model} : x \mapsto \hat{y} \in \mathbb{Y}$ zu entwickeln, das in der Lage ist, Eingabedaten

zuverlässig auf die korrekten Ausgaben abzubilden, und somit eine Generalisierung auf unbekannte Daten ermöglicht.

Zur Evaluierung der Performanz eines KNN wird der Datensatz in drei disjunkte Subsets segmentiert: Trainingsdaten ($\mathbb{X}train, \mathbb{Y}train$), Validierungsdaten ($\mathbb{X}val, \mathbb{Y}val$) und Testdaten ($\mathbb{X}test, \mathbb{Y}test$). Dabei wird ausschließlich das Trainingsdatenset während des Trainingsprozesses verwendet.

Unter Berücksichtigung der Funktionsausgabe eines MLP, wie in Gleichung 2.4 definiert, und einem Ensemble von Netzwerkparametern Θ, lässt sich die Ausgabe des MLP in Relation zu diesem Parametersatz neu formulieren. Dies wird in der folgenden Gleichung dargestellt:

$$\hat{y} = f_{Model}(x;\theta) = \varphi_n(f_n(...\varphi_2(f_2(\varphi_1(f_1(x_0))))) \tag{2.5}$$

Das primäre Ziel innerhalb eines maschinellen Lernkontextes ist es, für jede Probe $x^{(i)}$ im Trainingssatz einen Parametersatz θ zu ermitteln, der die Bedingung $\hat{y}^i \approx y^i$ hinreichend erfüllt, sodass die Modellausgabe eine zufriedenstellende Annäherung an den tatsächlichen Wert darstellt.

Um die Qualität des aktuellen Parametersatzes zu bestimmen, wird eine Zielfunktion $f_{Ziel}(x)$ definiert, die Beobachtungen aus einem gegebenen Datensatz vollständig auf ihre entsprechenden Labels abbildet. Mit einer Verlust- oder Kostenfunktion zum Beispiel der mittlere quadratische Fehler [2](engl.: Mean Squared Error (MSE)), erstmals eingeführt von Fisher et al. [35], kann ein Optimierungskriterium wie folgt definiert werden:

$$\mathcal{L}(\theta) = \frac{1}{|\mathcal{X}_{train}|} \sum_{x \in \mathbb{X}_{train}} \left[f_{Ziel}(x) - f_{Modell}(x;\theta) \right]^2 \tag{2.6}$$

Durch die Minimierung der Gleichung 2.6 wird die Vorhersagegenauigkeit des Lernalgorithmus, in diesem Beispiel KNNs, erhöht.

[2] Abhängig vom Lernproblem können auch andere Verlustfunktionen wie Cross Entropy Loss oder Likelyhood Loss angewendet werden.

Die Funktionswerte einer Verlustfunktion $\mathcal{L}$ in Bezug auf alle möglichen Parameterwerte bilden die sogenannte Verlustoberfläche. Das Optimierungsziel beim Training eines neuronalen Netzes ist demnach, das globale Optimum auf einer mehrdimensionalen Verlustfläche zu finden. Allerdings ist das Ermitteln eines globalen Optimums selbst für ein KNN mit nur einer geringen Anzahl von Neuronen ein NP-vollständiges Problem [36]. Daher werden KNN mit iterativen Methoden optimiert, die in der Lage sind, lokale Minima in der Verlustoberfläche zu ermitteln.

Gradientenabstiegsverfahren

Der Gradientenabstieg repräsentiert einen iterativen Optimierungsalgorithmus, der darauf abzielt, lokale Minima innerhalb des Definitionsraums einer gegebenen differenzierbaren Zielfunktion zu identifizieren. Innerhalb des Bereichs des maschinellen Lernens findet das Verfahren Anwendung, um eine Menge von Parametern hinsichtlich einer spezifischen Zielfunktion $\mathcal{L}$ zu optimieren. Dies erfolgt durch iterative Anpassungen der Parameterwerte basierend auf dem Gradienten der Zielfunktion $\nabla_\Theta \mathcal{L}$, wobei Θ die Menge der zu optimierenden Parameter symbolisiert. Im spezifischen Anwendungskontext des Trainings von KNN dient der Gradientenabstieg dem Zweck, eine Konfiguration von Gewichten und Biases für die Neuronen zu ermitteln, die den Fehler zwischen den Ausgaben des Netzes und den tatsächlichen Zielwerten minimiert. In jeder Iterationsphase des Optimierungsprozesses berechnet der Algorithmus den aktuellen Gradienten der Zielfunktion $\mathcal{L}$ bezüglich der Parametermenge Θ.

Die Eigenschaft des Gradienten einer Zielfunktion $\mathcal{L}$, den Vektor des steilsten Anstiegs im Funktionsraum zu repräsentieren, ist fundamental für die Funktionsweise des Gradientenabstiegsalgorithmus. Diese Richtungscharakteristik des Gradienten ermöglicht es, durch Aktualisierung der Modellparameter in der entgegengesetzten Richtung des Gradienten eine methodische Optimierung zu erzielen.Für die Durchführbarkeit dieses Verfahrens ist eine stetige Differenzierbarkeit der Aktivierungsfunktionen der Neuronen eine notwendige Voraussetzung.

Um das Risiko einer Überanpassung der Parameterwerte bei ihrer Aktualisierung zu minimieren, wird der berechnete Gradientenwert mit einem Skalierungsfaktor multipliziert, der als Lernrate bekannt ist. Diese Lernrate dient als Kontrollmechanismus, um die Schrittweite der Parameteraktualisierung zu

modulieren und somit eine stabile Konvergenz gegen das angestrebte Optimum zu fördern.

Die oben beschriebene Aktualisierungslogik des Gradientenabstiegs wird zusammenfassend im Algorithmus 1 dargestellt, welcher die schrittweise Vorgehensweise des konventionellen Gradientenabstiegsverfahrens illustriert.

Algorithmus 1 : Aktualisierungsregel des Gradientenabstiegs

Eingabe : $\mathcal{L}$, η
Result : Parameterwerte für Θ, die ein (lokales) Minimum von $\mathcal{L}$ beschreiben
Zufällige Initialisierung von Θ;
while *!(Abbruchkriterium erfüllt)* **do**
  $\Theta \leftarrow \Theta - \eta \cdot \nabla_\Theta \mathcal{L}(\Theta)$
end
Ausgabe : Θ

Das Gradientenabstiegsverfahren kann lokale Optima identifizieren, wodurch es gegebenenfalls das globale Optimum verfehlt und zu suboptimalen Lösungen führt. Die Auswahl einer angemessenen Lernrate η ist essenziell für die Effektivität des Trainingsprozesses. Zu niedrig angesetzte Werte von η können die Konvergenzgeschwindigkeit reduzieren und das Risiko erhöhen, dass das Modell in lokalen Optima verbleibt. Im Gegensatz dazu können überhöhte Werte von η zu Oszillationen um das Optimum oder gar zur Divergenz des Lernprozesses führen. In maschinellen Lernszenarien, insbesondere beim Training auf umfangreichen Datensätzen, stößt das Gradientenabstiegsverfahren an praktische Grenzen. Die Berechnung des Gradienten $\nabla_\Theta \mathcal{L}(\Theta)$ über den gesamten Datensatz ist rechenintensiv, was die Effizienz und Praktikabilität des Verfahrens in DL-Anwendungen einschränkt.

Über- und Unteranpassung von ML-Modellen

Die Konzepte der Überanpassung (Overfitting) und Unteranpassung (Underfitting) werden anhand von drei unterschiedlich konfigurierten MLPs veranschaulicht, die sich in der Anzahl ihrer verborgenen Schichten unterscheiden (siehe Abbildung 2.5). Die Visualisierung demonstriert die Auswirkungen auf die Modellgenauigkeit bei der Approximation einer verrauschten Sinuskurve. Die linke Grafik präsentiert ein MLP mit einer einzelnen verborgenen

Schicht (M = 1), welches eine unzureichende Anpassung an die zugrundeliegende Sinusfunktion zeigt. Die resultierende Approximationskurve weist eine deutliche Diskrepanz zu den tatsächlichen Datenpunkten auf, was charakteristisch für das Phänomen der Unteranpassung ist. Diese Konstellation führt zu einer verminderten Modellgenauigkeit, da das Modell nicht in der Lage ist, die Struktur der Daten adäquat zu erfassen. Im Kontrast dazu illustriert die rechte Grafik ein MLP mit zehn verborgenen Schichten. Hierbei ist zu beobachten, dass die Modellkurve die Datenpunkte inklusive des inhärenten Rauschens übermäßig genau nachbildet. Diese übertriebene Anpassung an die Trainingsdaten, bekannt als Überanpassung, kompromittiert die Fähigkeit des Modells, auf neuen, unbekannten Daten zu generalisieren. Als Folge dessen ist die Modellperformance außerhalb des Trainingsdatensatzes suboptimal.

Die mittlere Grafik stellt ein ausgewogenes Verhältnis dar, in dem ein MLP mit drei verborgenen Schichten verwendet wird. Diese Konfiguration ermöglicht eine präzise und zugleich generalisierbare Approximation der Sinusfunktion. Indem die Kurve die tatsächlichen Datenpunkte korrekt approximiert, ohne das inhärente Rauschen überzugewichten, wird ein Gleichgewicht zwischen Anpassungsfähigkeit und Generalisierungsfähigkeit erreicht. Dieses Modell verdeutlicht, wie durch eine angemessene Wahl der Modellkomplexität die Balance zwischen Über- und Unteranpassung effektiv gesteuert werden kann, um optimale Vorhersageleistungen auf unbekannten Daten zu erzielen.

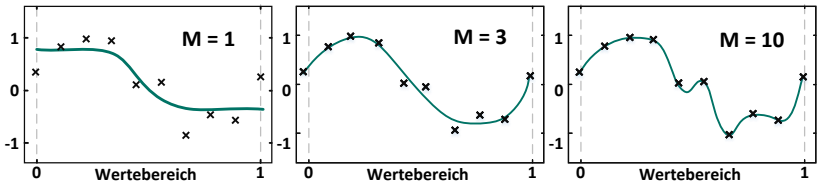

Abbildung 2.5: Visualisierung von Über- und Unteranpassungseffekten am Beispiel eines MLP beim Training auf eine verrauschte Sinuskurve. (nach [29])

2.2.3 Verifikation von Selbstlernenden Systemen

Nach der Aufbereitung der Daten und dem Training von ML-Modellen ist die Selektion des optimalen Modells mittels adäquater Leistungsmetriken wesentlich für die Entwicklung maschineller Lernfunktionen. Dies gilt für

Klassifikations- sowie Regressionsmodelle, welche jeweils anhand spezifischer Kriterien auf Basis eines getrennten Testdatensatzes bewertet werden, um Überanpassung (siehe Abbildung 2.5) zu verhindern. Ein entscheidender Schritt in diesem Prozess ist die Verifikation (siehe Gleichung 2.10) der Modelle, bei der überprüft wird, ob die implementierten Algorithmen und die erzielten Ergebnisse den spezifizierten Anforderungen und Zielvorgaben entsprechen.

Die Leistungsmetrik und die Verlustfunktion, diskutiert in Kapitel 2.2.2, erfüllen unterschiedliche Rollen im Kontext maschinellen Lernens. Verlustfunktionen quantifizieren den Fehler eines maschinellen Lernmodells und sind Bestandteil des Trainingsprozesses, der durch Optimierungsverfahren wie das Gradientenabstiegsverfahren gesteuert wird. Ihre Differenzierbarkeit ist dabei essenziell für die Anwendung dieser Optimierungstechniken. Leistungsmetriken hingegen dienen der Bewertung und Überwachung der Modellperformanz während Training und Evaluation und unterliegen keiner Differenzierbarkeitsanforderung. In bestimmten Fällen, wie beim mittleren quadratischen Fehler, kann eine Metrik aufgrund ihrer Differenzierbarkeit doppelt fungieren – sowohl als Verlustfunktion für das Training als auch als Leistungsmetrik für die Bewertung.

Die Genauigkeit einer Klassifizierung wird quantifiziert durch die Anzahl richtig positiver (true positives, TP) und richtig negativer (true negatives, TN) Ergebnisse, sowie durch die Anzahl falsch positiver (false positives, FP) und falsch negativer (false negatives, FN) Zuordnungen. Diese Werte konstituieren eine Konfusionsmatrix, welche eine visuelle Darstellung der Klassifizierungsleistung bietet, wie in Tabelle 2.1 am Beispiel der binären Klassifikation illustriert.

Tabelle 2.1: Konfusionsmatrix am Beispiel einer binären Klassifikation

		vorhergesagte Klasse (prediction)	
		positiv	**negativ**
korrekte Klasse (ground truth)	**positiv**	wahr positiv (true positivy, t_p)	falsch negativ (false negative, f_n)
	negativ	falsch positiv (false positiv, f_p)	wahr negativ (true negativ, t_n)

Bei einigen Anwendungen kann die Betrachtung einer einzigen Performanzmetrik zu einem unvollständigen Bild des Problems und damit zu einer falschen Interpretation der Ergebnisse führen. Anhand der Werte der Konfusionsmatrix lassen Metriken zur Bewertung von ML-Modellen zur Klassifikation ableiten. Metriken für ML-Modelle für Regressionsaufgaben werden in dieser Dissertation nicht betrachtet, sind aber in [37] zusammengefasst.

Eine etablierte Performanzmetrik ist die Genauigkeit der Vorhersage, bezeichnet durch:

$$Genauigkeit = \frac{t_p + t_n}{t_p + t_n + f_p + f_n} \tag{2.7}$$

mit der Anzahl der wahren positiven/negativen Vorhersagen t_p/t_n und der Anzahl der falsch positiven/negativen $f_p + f_n$ [38].

Je nach der Klassenverteilung des Datensatzes kann die Genauigkeitsmetrik jedoch irreführend sein und stellt keinen Indikator für die Leistung eines ML-Modells dar. Dieser Fall tritt auf, wenn die Klassenverteilung unausgewogen ist und eine Klasse häufiger vertreten ist als eine seltene Klasse.

Im Folgenden wird als Beispiel ein medizinischer Test für eine seltene Krankheit beschrieben. Es wird davon ausgegangen, dass nur eine Person von 500 000 Personen an dieser Krankheit erkrankt. Ein Klassifikator, z. B. in Form eines künstlichen neuronalen Netzes, der die Patienten immer als gesund einordnet, wird nach der obigen Formel 2.7 eine Genauigkeit von 99,9998 % erreichen. Somit lässt sich die Qualität eines solchen Systems nur unzureichend durch die Genauigkeit charakterisieren.

Eine Möglichkeit, dieses Phänomen zu lösen, besteht darin, stattdessen die Precision und den Recall zu ermitteln. Die Precision berechnet sich entsprechend der Gleichung 2.8 und gibt den Anteil an richtig vorhergesagten positiven Ergebnissen (t_p) bezogen auf die Gesamtheit aller als positiv vorhergesagten Ergebnisse ($t_p + f_p$) an.

$$Precision = \frac{\mathrm{t_p}}{\mathrm{t_p} + \mathrm{f_p}} \tag{2.8}$$

Der Recall (siehe Gleichung 2.9), oder die Trefferquote, stellt den Anteil der korrekt als positiv eingeordneten Testergebnisse (t_p) im Verhältnis zur Gesamtzahl der tatsächlich positiven Ergebnisse ($t_p + f_p$) dar.

$$Recall = \frac{\mathrm{t_p}}{\mathrm{t_p} + \mathrm{f_n}} \tag{2.9}$$

Ein maschineller Lernalgorithmus, der stets vorhersagt, dass keine Person erkrankt ist, würde eine perfekte Präzision, jedoch einen Recall von 0 erzielen. Im Gegensatz dazu würde ein Algorithmus, der bei jeder getesteten Person das Vorliegen der Krankheit diagnostiziert, einen perfekten Recall erreichen, während die Präzision dem Anteil der tatsächlich erkrankten Personen in der Population entspricht.[3].

In maschinellen Lernmodellen gibt es einen Schwellenwertparameter, der die Balance zwischen Präzision und Recall beeinflusst. Eine Erhöhung des Schwellenwerts kann die Präzision erhöhen, führt jedoch in der Regel zu einem Rückgang des Recalls. Durch die gezielte Anpassung des Schwellenwerts kann somit der gewünschte Precision-Recall-Trade-off definiert werden. Abbildung 2.6 veranschaulicht die Abhängigkeit von Präzision und Recall in Bezug auf den Schwellenwert bei einem KNN-Modell.

Die Bestimmung des optimalen Schwellenwerts ist stark an den spezifischen Anwendungsfall gebunden. Modelle, die einen hohen Recall erfordern – beispielsweise ab einem Wert von 0,8 oder 80% –, sind in Bereichen wie der Krankheitserkennung von großer Bedeutung, wo falsch-negative Ergebnisse schwerwiegende Folgen haben können. In solchen Fällen wird eine hohe Abdeckung von falsch-negativen Vorhersagen bevorzugt, da es kritischer ist, eine Krankheit zu übersehen, als eine gesunde Person fälschlicherweise als krank zu klassifizieren.

Im Gegensatz dazu ist in Anwendungsfeldern wie Empfehlungssystemen oder der Spam-Filterung eine hohe Präzision wünschenswert, üblicherweise ab einem Wert von 0,8 oder 80%. In diesen Kontexten ist es weniger relevant, falsch-negative Fälle zu erfassen, während die Minimierung falsch-positiver

[3] 0,0001 Prozent in diesem Beispiel, bei einer Krankheit, die nur einer von einer Million Menschen hat

Vorhersagen im Vordergrund steht. Eine Präzision von 80% bedeutet, dass die Mehrheit der als positiv klassifizierten Vorhersagen korrekt ist, was für diese Anwendungsfälle entscheidend ist.

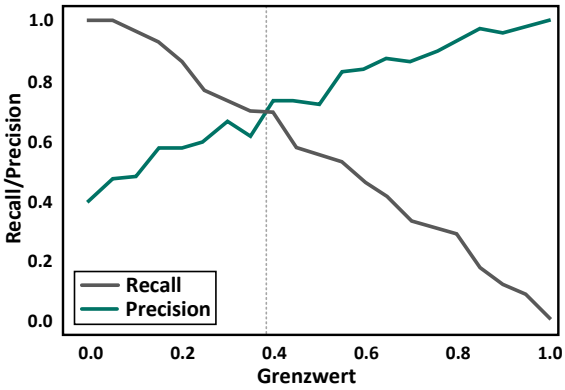

Abbildung 2.6: Visualisierung des Precision- und Recall-Kurve über den Grenzwert

Der Schnittpunkt zwischen der Precision- und der Recall-Kurve wird als Break-even-Punkt bezeichnet. Es existieren jedoch Anwendungsfälle, bei denen sowohl die Precision als auch der Recall relevant sind. Daher wurde der F1-Score eingeführt, der eine Möglichkeit darstellt, Precision und Recall in einer einzigen Metrik zu kombinieren. Der F1-Score, der als harmonisches Mittel von Präzision und Recall bezeichnet wird, ist wie folgt definiert:

$$\text{F1-Score} = \left(\frac{1/\text{ Precison } + 1/\text{ Recall}}{2} \right)^{-1} \tag{2.10}$$

Der F1-Score (Gleichung 2.10) wird verwendet, um die Leistung von zwei Klassifikatoren gegeneinander zu vergleichen. Angenommen, es wurden zwei Klassifikatoren trainiert. Ein Klassifikator verfügt über eine höhere Trefferquote und der zweite Klassifikator über eine höhere Genauigkeit. In diesem Fall können die F1-Werte für beide Klassifikatoren verwendet werden, um festzustellen, welcher Klassifikator bessere Ergebnisse liefert.

2.3 Vorgehensmodelle und Testmethoden

Seit der Industrialisierung werden Prozesse und Modelle zur effizienten Produktentwicklung eingesetzt. Effizienz bedeutet in diesem Kontext der optimale Einsatz von finanziellen, materiellen und personellen Ressourcen, die durch betriebswirtschaftliche Rahmenbedingungen begrenzt sind. Frederick Winslow Taylor[4] beschrieb in seinem Werk „Die Grundsätze wissenschaftlicher Betriebsführung“ Aspekte der modernen Prozessgestaltung, die die Grundlage für die Planung industrieller Prozesse bilden. Das Ziel produzierender Unternehmen ist laut [39]

> ...die Entwicklung, Herstellung und der Vertrieb marktfähiger Produkte mit dem Ziel der Gewinnmaximierung.

Um dieses Ziel zu erreichen, wird die Produktentwicklung durch Vorgehensmodelle (siehe Definition 2.6) gesteuert, die sich zyklisch wiederholen. Diese Modelle dienen als „Schablone“ oder Muster und sind nach dem Standard ISO/IEC 12207 [40] wie folgt definiert:

> **Definition 2.6 Vorgehensmodell:** Ein Vorgehensmodell bezeichnet ein Rahmenkonzept von Prozessen und Aktivitäten, die sich mit dem Lebenszyklus befassen, in Stufen organisiert sein können und als gemeinsame Referenz für die Kommunikation und das Verständnis dienen. [40]

Systementwicklungsprozesse wie das Wasserfallmodell nach Royce [41], das V-Modell nach Boehm [42] und das Spiralmodell nach Calvez [43] werden als traditionelle Software-Entwicklungsmethoden bezeichnet [44].

Die zunehmende Vernetzung der Bauteile in der Automobilindustrie erfordert eine ganzheitliche Betrachtung der Produktentstehung. Dies bedeutet, dass alle Aspekte des Produkts, von der Entwicklung der einzelnen Komponenten bis hin zur Integration und Vernetzung der Systeme, in einem Gesamtkontext betrachtet werden müssen. Das V-Modell 97 ist ein sequenzielles Vorgehensmodell

[4] Frederik Winslow Taylor, US-Amerikaner, 1856-1915, Begründer des Prinzips des Taylorismus, Publizierte 1911 sein Opus Magnum „The Principles of Scientific Management“ und führte 1913 die Fließbandproduktion zusammen mit Henry Ford ein

für die Systementwicklung, das speziell auf die Bedürfnisse der öffentlichen Verwaltung angepasst wurde. Es basiert auf dem generischen V-Modell und unterteilt den Entwicklungsprozess in vier Submodule [45]:

- Systemerstellung (SE): Dies umfasst die Kernentwicklungsphasen, die sich mit der eigentlichen Erstellung des Systems befassen.
- Qualitätssicherung (QS): Dieses Modul stellt sicher, dass das System über den gesamten Entwicklungsprozess hinweg die definierten Qualitätsanforderungen erfüllt.
- Konfigurationsmanagement (KM): Dieses Modul verwaltet alle Konfigurationen des Systems. Es dokumentiert Änderungen, verwaltet Versionen und sorgt dafür, dass die einzelnen Konfigurationen nachvollziehbar bleiben.
- Projektmanagement (PM): Dieses Modul umfasst alle Aufgaben und Maßnahmen, die für die Planung, Steuerung und Kontrolle des Projekts erforderlich sind.

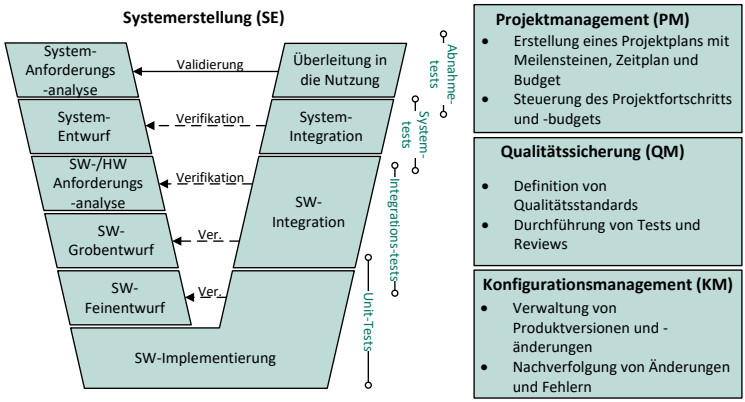

Abbildung 2.7: Das V-Modell 97: Ein Vorgehensmodell für die Softwareentwicklung in der Bundesverwaltung (nach [45])

Das V-Modell, eine Erweiterung des Wasserfallmodells, bildet den Kern des Submoduls Systemerstellung (SE). Es stellt einen strukturierten, zweigeteilten Entwicklungszyklus dar, der die systematische Analyse und Umsetzung von

Kundenanforderungen bis zum akzeptanzgetesteten Endprodukt beschreibt. Einerseits werden die Systemanforderungen in einem Top-down-Ansatz definiert und auf verschiedene Abstraktionsebenen heruntergebrochen, um eine Verbindung zwischen System- und Komponentenebene herzustellen. Andererseits erfolgt die Integration und das Testen der entwickelten Komponenten in einem Bottom-up-Ansatz, wobei jede Phase sicherstellt, dass sowohl Einzelteile als auch das Gesamtsystem den Anforderungen entsprechen. Dieses Vorgehen ermöglicht eine systematische Überprüfung auf jeder Entwicklungsstufe, fördert die Effizienz und Effektivität der Produktentwicklung und führt zu einem qualitativ hochwertigen und zuverlässigen Endprodukt [45].

Ferner erleichtert das V-Modell die Zusammenarbeit zwischen Automobilherstellern und Systemlieferanten, indem es die Verantwortlichkeiten abgrenzt und definierte Schnittstellen für Spezifikationen und Integrationen vorgibt.

Definition 2.8 Anforderung: Ein Kriterium oder eine Eigenschaft, die von einem System oder einer seiner Komponenten erfüllt oder aufgewiesen werden muss, um vertragliche Vereinbarungen, Standards, Spezifikationen oder Dokumentationen zu entsprechen.

Die Anwendung des V-Modells setzt voraus, dass die Kunden- und Nutzeranforderungen zu Beginn eines Projekts vollständig und korrekt erfasst sind. Der sequenzielle Aufbau des Modells garantiert nur unter dieser Bedingung einen effizienten Prozess, da Unstimmigkeiten in den Spezifikationen erst bei der Integration aufgedeckt werden.

In der Praxis sind die Anforderungen jedoch zu Beginn nicht vollständig bekannt und werden während des Projekts präzisiert. Um dies zu berücksichtigen, kann das V-Modell auf verschiedene Weise angepasst werden:

- Iteratives oder inkrementelles Vorgehen: Das Projekt wird in mehreren Iterationen oder Inkrementen durchgeführt, wobei in jeder Iteration ein Teil der Anforderungen umgesetzt wird. Dies ermöglicht eine schrittweise Anpassung der Anforderungen an die Bedürfnisse der Kunden und Nutzer.
- Multi-Phasen-Implementierung: Das V-Modell wird mehrmals durchlaufen, wobei in jeder Phase ein Teil der Anforderungen umgesetzt wird.

Dies ermöglicht eine Anpassung des Modells an die spezifischen Bedürfnisse des Projekts.

Eine einphasige Implementierung des V-Modells ist nach [46] unter bestimmten Bedingungen ratsam:

- Alle Anforderungen sind von Beginn an bekannt und stabil.
- Der optimale Lösungsansatz kann aus allen bekannten Alternativen ausgewählt werden.
- Die ausgewählte Lösung ist durchführbar.
- Es wird eine „first-time-right“-Lösung für das gesamte System angenommen.

Um die Erfüllung der Anforderungen zu gewährleisten, erfolgt eine Unterscheidung zwischen Verifikation und Validierung. Für eine spezifisch auf die Automobilentwicklung bezogene Erläuterung dieser Konzepte bietet Schäuffele [47] eine Definition.

Definition 2.10 Verifikation: Verifikation bezeichnet den Vorgang der Bewertung eines Systems oder einer Komponente, um zu überprüfen, ob die Ergebnisse einer bestimmten Entwicklungsstufe den festgelegten Anforderungen dieser Stufe gerecht werden. Bei der Software-Verifikation geht es somit um die Überprüfung, ob die Umsetzung den spezifischen Vorgaben des jeweiligen Entwicklungsschritts entspricht. [47]

Definition 2.12 Validierung: Validierung ist der Bewertungsprozess eines Systems oder einer Komponente mit dem Zweck, zu entscheiden, ob es für den beabsichtigten Einsatz geeignet ist oder die Erwartungen der Nutzer trifft. Bei der Funktionsvalidierung wird untersucht, inwiefern eine Spezifikation den Anforderungen der Benutzer entspricht und ob eine Funktion die Zustimmung der Anwender findet. [47]

Das V-Modell bietet eine strukturierte Methode für die Systementwicklung, beschreibt jedoch nicht im Detail, welche Testmethoden in den einzelnen Phasen

eingesetzt werden sollten. Die Auswahl der Testmethoden hängt von verschiedenen Faktoren ab, wie z.B. der Art des Systems, den Anforderungen und dem Risiko.

> **Definition 2.14 Testen:** Aktivität, bei der ein System oder eine Komponente unter bestimmten Bedingungen ausgeführt wird, die Ergebnisse beobachtet oder aufgezeichnet werden und eine Bewertung gegen die erwarteten Anforderungen des Systems oder der Komponente vorgenommen wird [48].

Das Testen von Software stellt eine zentrale Technik zur Verifizierung und Validierung der Qualität von Softwarefunktionen dar (siehe Definition 1.2) [49]. Der Testprozess ermöglicht den Entwicklern, Vertrauen in ein System aufzubauen, indem er die Abwesenheit von Fehlverhalten überprüft (siehe Kapitel 1). Softwaretests werden gezielt durchgeführt, um Programme oder Systeme mit dem Ziel der Fehleridentifikation zu evaluieren [50]. Dieser Prozess ist sowohl arbeitsintensiv als auch kostenaufwendig, insbesondere aufgrund der Notwendigkeit, spezifische Testfälle zu erstellen (siehe Definition 1.4). Im Automobilsektor entfallen mehr als 50% der Gesamtkosten der Softwareentwicklung auf Testaktivitäten [51].

> **Definition 2.16 Fehler:** Ein Fehler ist die Nichterfüllung (Nichtkonformität) einer festgelegten Anforderung. Der Fehler erzeugt somit eine Abweichung zwischen den vorhandenen und den definierten Anforderungen (siehe Definition 2.8) . Unter Anforderungen können zum Beispiel Qualitäts- und Zuverlässigkeitsmerkmale, Produktanforderungen, Kundenforderungen verstanden werden. [52]

Black-Box-Tests und White-Box-Tests sind zwei grundlegende Testmethoden, die in jeder Phase des V-Modells (siehe Abbildung 2.7) eingesetzt werden können. Im Kontext des V-Modells spielen Black-Box- und White-Box-Tests als Testmethoden eine zentrale Rolle [53].

Black-Box-Tests simulieren die Nutzung der Software aus der Perspektive des Benutzers, ohne Kenntnis der internen Code-Struktur. Der Fokus liegt auf der Prüfung der Schnittstellen, der Eingabe-Ausgabe-Beziehungen und der resultierenden Systemfunktionalität. White-Box-Tests hingegen fokussieren sich auf die interne Code-Struktur und zielen auf die Codeabdeckung, die Qualität

des Codes und die zugrundeliegende Logik ab. Diese Testmethode ermöglicht eine tiefgreifende Analyse der Softwareimplementierung.

Die Verknüpfung des V-Modells mit Black-Box- und White-Box-Tests kann auf verschiedene Weise erfolgen (siehe Abbildung 2.7):

1. **Zuordnung der Testmethoden zu den V-Modell-Phasen**

    - Unit-Tests: In dieser Phase kommen vorwiegend White-Box-Tests zum Einsatz, um die einzelnen Module der Software isoliert zu testen.
    - Integrationstests: Sowohl Black-Box- als auch White-Box-Tests können in dieser Phase verwendet werden, um die Interaktionen zwischen den Modulen zu testen und die Kohäsion der Software zu verbessern.
    - Systemtests: Black-Box-Tests dominieren diese Phase, um die Systemfunktionalität aus der Benutzerperspektive zu evaluieren.
    - Abnahmetests: In der finalen Phase können beide Testmethoden eingesetzt werden, um die Akzeptanz des Systems im Hinblick auf die spezifischen Anforderungen des Kunden zu beurteilen.

2. **Auswahl der Testmethoden anhand des Testziels:**

    - Black-Box-Tests: Geeignet für die Prüfung der Systemfunktionalität aus der Benutzerperspektive, z.B. Funktionstests, Usability-Tests, Kompatibilitätstests und Performance-Tests.
    - White-Box-Tests: Geeignet für die tiefgreifende Analyse der Code-Struktur, z.B. Unit-Tests, Code-Reviews, statische Codeanalysen und Mutationstests.

2.3.1 Szenariobasiertes Testen

Die zunehmende Automatisierung von Kraftfahrzeugen erfordert eine stringente Sicherstellung des Systemverhaltens. Das szenariobasierte Testen etabliert sich in diesem Kontext als effiziente Methode zur Validierung der funktionalen und sicheren Eigenschaften autonomer Fahrsysteme.

Mit steigendem Automatisierungsgrad gemäß der ISO/SAE 22736 Norm [54] stoßen distanzbasierte Testansätze an ihre Grenzen. Die ökonomische Rentabilität dieser Verfahren sinkt, da die Anzahl der zu testenden Anforderungen exponentiell wächst. Die Veröffentlichung von Bagschik et al. [55] adressiert diese Herausforderung und schlägt einen szenariobasierten Ansatz vor, der auf der Norm ISO 26262 [56] basiert. Diese Norm definiert einen Entwicklungsprozess für sicherheitskritische, elektrische und elektronische Systeme in Kraftfahrzeugen, einschließlich Fahrerassistenzsystemen.

Szenariobasiertes Testen ist eine Testmethode, die sich auf die Validierung des Systemverhaltens unter realistischen Bedingungen und aus der Perspektive der Endbenutzer konzentriert. Im Gegensatz zu anderen Testmethoden, die sich auf die interne Struktur des Systems fokussieren, betrachtet das szenariobasierte Testen das System als Black Box und testet es anhand von vordefinierten Szenarien.

Der szenariobasierte Testprozess umfasst folgende Phasen:

- Identifikation von Anwendungsfällen: Relevante Anwendungsfälle für die zu testende Software werden definiert.
- Erstellung von Testszenarien: Für jeden Anwendungsfall wird ein Testszenario erstellt, welches die Interaktionen zwischen System und Benutzer abbildet.
- Priorisierung der Testszenarien: Die Szenarien werden nach Relevanz und Risiko priorisiert.
- Durchführung der Tests: Die Testszenarien werden manuell oder automatisiert ausgeführt.
- Auswertung der Ergebnisse: Die Testergebnisse werden analysiert und Fehler dokumentiert.

Die ISO-Norm 26262 [56] verwendet Szenarien, um die Arbeitsergebnisse der einzelnen Prozessschritte im V-Modell zu generieren. Diese Definition von Szenarien stammt von Ulbrich et al. [57].

Definition 2.18 Szenario: Ein Szenario beschreibt den zeitlichen Ablauf und die Entwicklung einer Geschichte. Es beinhaltet die vorbestimmte Reihenfolge mehrerer Szenen und die dazugehörigen Situationen der Teilnehmer in der jeweiligen Szenerie.

Jeder Prozessschritt im V-Modell liefert Arbeitsergebnisse, die durch Szenarien (siehe Definition 2.18) mit unterschiedlichen Abstraktionen erreicht werden. Dies dient der Gewährleistung der funktionalen Sicherheit. Die Anforderungen an die Szenarien variieren jedoch je nach Entwicklungsschritt. Daher schlagen Ulbrich et al. [57] vor, Szenarien auf unterschiedlichen Abstraktionsebenen zu definieren, die ineinander umgewandelt werden können.

Jedes dieser Szenarien hat einen unterschiedlichen Grad an Konkretisierung der Informationen. Die Abstraktion nimmt in jeder Ebene ab, während die Anzahl der möglichen Szenarien ansteigt.

Funktionale Szenarien

Funktionale Szenarien stellen ein grundlegendes Konzept der Systementwicklung dar. Sie repräsentieren abstrakte Beschreibungen des Systems aus der Perspektive der Nutzer und dienen somit der Verdeutlichung von dessen Funktionen und Anforderungen. Die Szenariobeschreibung spezifiziert sowohl die Systemfunktionalität selbst als auch den Kontext ihrer Nutzung.

Die Verwendung funktionaler Szenarien ist ein integraler Bestandteil der Gefährdungsanalyse. Durch die frühzeitige Identifikation von Risiken und potenziellen Gefährdungen in der Szenariobeschreibung können Probleme bereits in der Planungsphase behoben werden. Dies wirkt präventiv gegen spätere Komplikationen im Entwicklungsprozess. Funktionale Szenarien befinden sich auf der obersten Abstraktionsebene und werden in sprachlicher Form, d.h. natürlicher Sprache, repräsentiert. Diese Darstellungsform zeichnet sich durch ihre Verständlichkeit und intuitive Lesbarkeit aus. Das Abstraktionsniveau erlaubt es, funktionale Szenarien auf einer Bandbreite von relevanten Situationen abzubilden. [58]

Die Definition funktionaler Szenarien bildet die Basis für die Erzeugung von Szenarien. Diese Szenarien können durch voneinander abgegrenzte Begriffe oder Entitäten beschrieben werden, die miteinander in Beziehung stehen

und interagieren können. Zur Vermeidung von Missverständnissen und zur Sicherstellung einer effizienten Systementwicklung ist die Verwendung einer einheitlichen Terminologie für die verwendeten Begriffe von zentraler Bedeutung. Beispielsweise kann die Beschreibung der Umgebung durch die Begriffe " ´Sommer“ und " ´Regen“ erfolgen, um die Wetterbedingungen zu definieren.

Logische Szenarien

Logische Szenarien werden aus funktionalen Szenarien abgeleitet und bilden somit die zweite Abstraktionsebene in der Systementwicklung. Die funktionalen Szenarien beschreiben die Systemfunktionalität aus der Nutzerperspektive, während die logischen Szenarien die Darstellung der Systemlogik auf einer abstrakteren Ebene ermöglichen.

Die Grundlage der logischen Szenarien bildet die Darstellung der Entitäten im physikalischen Zustandsraum. Jede Entität ist durch einen Satz von Parametern charakterisiert, die ihren Zustand beschreiben. Die Wertebereiche dieser Parameter spezifizieren die möglichen Zustände der jeweiligen Entität. Beispielsweise könnte der Wertebereich für die Temperatur in Grad Celsius oder für die Tröpfchengröße in Mikrometer definiert sein.

Logische Szenarien differenzieren zwischen absoluten und relativen Parametern. Absolute Parameter beschreiben die Eigenschaften der Entitäten selbst, während relative Parameter die Beziehungen und Korrelationen zwischen den Entitäten abbilden. Diese Korrelationen können durch numerische Bedingungen oder Korrelationsfunktionen formalisiert werden. Die Parameter der logischen Szenarien können mit Wahrscheinlichkeitsaussagen versehen werden. Diese werden durch statistische Verteilungen, wie z.B. die Normalverteilung oder Gleichverteilung, beschrieben. Die Berücksichtigung von Wahrscheinlichkeitsverteilungen ermöglicht die Modellierung von Unsicherheiten in den Parametern und deren Auswirkungen auf das Systemverhalten.

Konkrete Szenarien

Die konkrete Szenarienebene stellt die abschließende Abstraktionsebene in der Systementwicklung dar. Sie beschreibt die Entitäten des Systems mit höchster Präzision und definiert ihre Parameter eindeutig. Diese Ebene wird aus den logischen Szenarien abgeleitet, indem für jeden Parameter ein konkreter Wert

aus seinem Wertebereich festgelegt wird. Im Kontext der Testabdeckung ermöglicht die infinitesimale Abtastung der kontinuierlichen Wertebereiche die Generierung von theoretisch unendlich vielen Testfällen. Durch diese umfassende Testabdeckung kann die Funktionalität und Zuverlässigkeit des Systems in hohem Maße sichergestellt werden.

Die Menge an generierten Testfällen kann jedoch aufgrund der Anzahl unpraktikabel sein. In der Praxis ist es daher notwendig, die Anzahl der Testfälle zu reduzieren. Dies kann durch die Auswahl eines Experten oder durch eine Priorisierung anhand von Leistungsmetriken erfolgen [59].

Gemäß der Quelle von [58] müssen für die Durchführung von Tests folgende Anforderungen an die konkreten Szenarien erfüllt sein:

- Die Szenarien müssen eindeutig definiert sein und dürfen keine Interpretationsmöglichkeiten aufweisen, um Reproduzierbarkeit zu gewährleisten.
- Insbesondere im Kontext des Testorakelproblems (siehe 1.14), bei dem ein Testorakel (siehe 1.12) die erwartete Ausgabe für einen gegebenen Eingabeparameter bereitstellt, müssen korrekte und inkorrekte Testfälle eindeutig definiert werden.
- Die Darstellung der Szenarien muss effizient, maschinenlesbar sein.

In den ersten beiden Abschnitten dieser Dissertation (siehe Abschnitt 6 und 7) wird die Identifikation funktionaler und die Definition logischer Szenarien behandelt. Diese Szenarien befinden sich auf der höchsten Abstraktionsebene und dienen als Ausgangspunkt für die Ableitung konkreter Szenarien und die Generierung von Testdaten (siehe Abschnitt 7).

2.3.2 Szenariodefinition und morphologische Matrix

Das Forschungsprojekt PEGASUS konzipierte ein sechsstufiges Ebenenmodell, das zur Beschreibung von Szenarien Anwendung findet (siehe Abbildung 2.8). Dieses Modell nutzt unterschiedliche Ebenen, um die Aspekte und Charakteristika eines Szenarios abzubilden. Im Gegensatz zur Definition von Ulbrich et al. [57] liegt der Fokus des sechsstufigen Ebenenmodells [60] auf

einer Darstellungsform, die beispielsweise für Simulationen herangezogen werden kann.

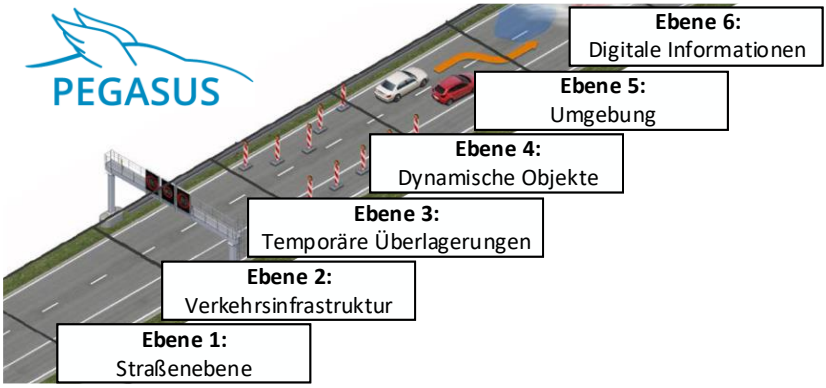

Abbildung 2.8: Darstellung des 6-Ebenen-Modells zur Beschreibung von Szenarien nach [60]

- **Ebene 1 und 2**: Die ersten beiden Ebenen des Modells definieren die Grundlage des Szenarios. Sie umfassen die Straßentopologie (z.B. Kurven, Geraden, Steigungen), die Beschaffenheit der Fahrbahn (z.B. Asphalt, Beton, Schotter) und die Leitinfrastruktur (z.B. Begrenzungen, Markierungen, Verkehrszeichen).
- **Ebene 3**: Temporäre Überlagerungen: Die dritte Ebene beschreibt temporäre Veränderungen von Ebene 1 und 2. Dazu zählen z.B. geänderte Spurführungen, Baustellen oder umgeleitete Verkehrsführungen.
- **Ebene 4**: Bewegliche Objekte: Die vierte Ebene fokussiert sich auf bewegliche Objekte im Szenario. Hierzu gehören andere Verkehrsteilnehmer (z.B. Autos, Fahrräder, Fußgänger) und deren manöverbasierte Interaktion (z.B. Überholvorgänge, Spurwechsel).
- **Ebene 5**: Umweltbedingungen: Die fünfte Ebene befasst sich mit den Umweltbedingungen, die das Szenario beeinflussen. Dazu zählen z.B. Witterung (z.B. Regen, Schnee, Nebel), Lichtverhältnisse (z.B. Tag, Nacht, Dämmerung) und Sichtbedingungen (z.B. eingeschränkte Sicht durch Nebel oder Regen).

- **Ebene 6**: Digitale Informationen: Die sechste und letzte Ebene umfasst digitale Informationen, die für das Szenario relevant sind. Hierzu zählen z.B. Vehicle-to-Everything (V2X)-Kommunikation (z.B. Informationen über andere Verkehrsteilnehmer) und digitale Karten (z.B. hochauflösende Karten mit Fahrbahnmarkierungen und Verkehrszeichen).

Szenariodefinition durch die Verwendung der morphologischen Matrix

Die Definition von logischen Szenarien auf der Grundlage von Expertenwissen ermöglicht die Erstellung von Szenarien, die den Anforderungen und Zielen des Testprozesses gerecht werden. Die Nutzung von Expertenwissen für die Definition von Szenarien stößt auf Grenzen, da die Bandbreite potenzieller Szenarien nicht durch Expertenwissen abgedeckt werden kann. Es existiert das Risiko, dass bestimmte Szenarien übersehen werden oder unvorhergesehene Ereignisse auftreten, die außerhalb des Erfahrungsbereichs der Experten liegen.

Diese Einschränkung kann die Effektivität und Vollständigkeit der Szenariodefinition beeinträchtigen, da relevante oder kritische Szenarien unberücksichtigt bleiben, wodurch Lücken in der Testabdeckung auftreten können. Um dieses Problem zu adressieren, können verschiedene Ansätze verwendet werden. Ein Ansatz besteht darin, mehrere Experten einzubeziehen, um eine breitere Perspektive und Einsichten zu erhalten. Die Zusammenarbeit in Form von Workshops oder Diskussionsrunden kann dazu beitragen, unterschiedliche Standpunkte und Erfahrungen zu berücksichtigen und somit die Wahrscheinlichkeit zu erhöhen, relevante Szenarien zu identifizieren.

> **Definition 2.20 morphologische Matrix:** Eine morphologische Matrix ist eine analytische Kreativtechnik zur strukturierten Erfassung von Informationen und systematischen Lösungserarbeitung. Entwickelt von Fritz Zwicky [61] im Jahr 1967, besteht sie aus einer Tabelle mit Kategorien von Merkmalen, die für ein Problem relevant sind. Jede Kategorie enthält mehrere Lösungsvarianten, deren Kombination neue Lösungen ermöglicht.

Donker et al. [62] haben die Erstellung einer morphologischen Matrix sowie die notwendigen Prozessschritte beschrieben (siehe Abbildung 2.9).

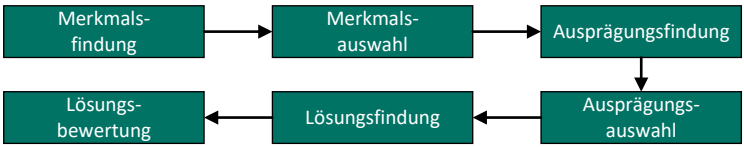

Abbildung 2.9: Prozessschritte zur Erstellung einer morphologischen Matrix. (Modifizierte Darstellung in Anlehnung an [62]).

Das Verfahren zur Erstellung einer morphologischen Matrix beginnt mit der Definition des Problems und der Vereinfachung dieses Problems. Anschließend werden Faktoren aufgelistet, die Einfluss auf die Lösungen des Problems haben. Diese Faktoren werden im Folgenden als Merkmale bezeichnet. Merkmale müssen einerseits differenzierbar und andererseits generalisierend gehalten werden, sodass die Problemstellung möglichst vollständig mit den genannten Merkmalen abgebildet werden kann.

Im nächsten Schritt werden für jedes Merkmal unterschiedliche Ausprägungen bzw. Realisierungsmöglichkeiten aufgelistet, die klar definiert und voneinander abgegrenzt sind. Diese Merkmale und ihre Ausprägungen werden in einer morphologischen Matrix zusammengefasst, die als Grundlage zur systematischen Kombination verschiedener Lösungsmöglichkeiten dient. Anschließend erfolgt eine Analyse der resultierenden Kombinationen, wobei auf nicht realisierbare Lösungen hingewiesen wird. Die realisierbaren Lösungen werden anhand von Bewertungskriterien geprüft, um die besten Alternativen zu identifizieren. Durch die strukturierte Vorgehensweise der Matrix lassen sich realistische Szenarien mit unterschiedlichen Ausprägungen entwickeln. Dieser Ansatz ermöglicht eine methodische Herangehensweise zur Erstellung von Szenarien für selbstlernende Funktionen mit Benutzeraktionen, wobei die morphologische Matrix als Leitfaden für die Kombination der Merkmale dient. (siehe Unterabschnitt 6.2.2).

3 Klassifikation und Analyse von Testorakeln in der Softwareentwicklung

Ein Testorakel[1] (siehe Definition 1.12) wird in der Softwareentwicklung für die Validierung von Softwarefunktionen (siehe Definition 1.2) eingesetzt.

In seinem 1970 erschienenen Artikel „Notes on Structured Programming“ bezeichnet Dijkstra [63] den Begriff „Orakel“, um sich auf eine externe Quelle zu beziehen, die die korrekten Ergebnisse eines Programms für einen bestimmten Satz von Eingabedaten liefert. Traditionelles Testen, beispielhaft vertreten durch das V-Modell (siehe Abbildung 2.7) setzt das Vorhandensein eines Testorakels voraus, mit dem die Testergebnisse anhand der Anforderungen der Entwickler an das System durch Testanforderungen spezifiziert und durch die Testergebnisse bestätigt werden können.

Barr et al. [64] beschreiben drei Hauptkategorien von Testorakeln, die jeweils spezifische Eigenschaften und Einsatzbereiche besitzen. Diese Kategorien bieten auf unterschiedliche Weise ein Testorakel an, indem sie verschiedene Methoden zur Generierung von Testfällen und Vorhersagen nutzen. Der Testorakelprozess (siehe Abbildung 3.1) besteht darin, dass eine Testeingabe sowohl dem Testorakel als auch dem System under Test (SuT) vorgelegt wird. Anschließend werden die Ergebnisse der beiden Entitäten miteinander verglichen, um die Korrektheit des SuT zu validieren.

[1] In der Softwareentwicklung wird der Begriff „Orakel“ metaphorisch verwendet, um eine Informationsquelle zu beschreiben, die als „allwissend“ in Bezug auf das erwartete Verhalten einer Funktion angesehen wird.

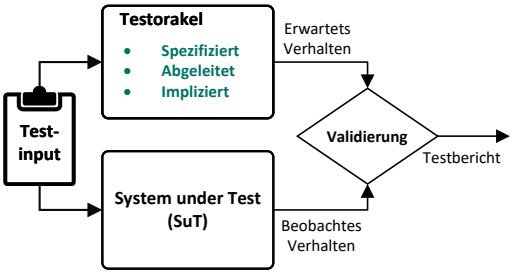

Abbildung 3.1: Illustration des Testorakelprozesses für spezifizierte, abgeleitete und implizierte Testorakel für die Validierung von Software

3.1 Spezifizierte Testorakel

Ein spezifiziertes Testorakel (STO) ist in der Softwareentwicklung eine definierte Entität, die eine formale oder semiformale Beschreibung dessen liefert, was als korrektes Verhalten einer Software unter bestimmten Testbedingungen betrachtet wird. Sie basieren auf den vorab festgelegten Anforderungen, Spezifikationen oder dem erwarteten Verhalten des zu testenden Systems. Ein STO ermöglicht die Bewertung der Testergebnisse, indem es Kriterien vorgibt, gegen die die tatsächlichen Ausgaben oder Zustände der Software verglichen werden können. Die Entwicklung STO erfordert jedoch einen initialen Aufwand, da für jedes Testziel individuelle Spezifikationen ausgearbeitet werden müssen, was sowohl Systemwissen der Anforderungen voraussetzt. STO lassen sich weiter in modell- oder zustandsbasierte und übergangsbasierte Spezifikationssprachen unterteilen.

3.1.1 Modellbasierte Spezifikationssprachen

Modellbasierte Spezifikationssprachen beschreiben ein System als eine Kombination von Zuständen und Operationen, die diese Zustände verändern. Deshalb werden sie in der Fachliteratur [65, 66] auch als zustandsbasierte Spezifikationen bezeichnet. Die Operationen des Systems werden durch verschiedene Bedingungen, sogenannte Vorbedingungen und Nachbedingungen, eingeschränkt. Eine Vorbedingung definiert eine notwendige Anforderung an die Eingabe, die erfüllt sein muss, damit die Operation korrekt ausgeführt werden

kann. Eine Nachbedingung definiert die Auswirkung, die die Operation auf den Programmzustand hat [66].

Ein Beispiel für modellbasierte Spezifikationssprachen ist die Z-Notation [67], die in der Modellierung und Spezifikation von Computersystemen zum Einsatz kommt und sich durch mathematische Präzision sowie die Nutzung von Mengenlehre und Logik hervorhebt. Object-Z [68] ist eine Erweiterung der Z-Notation, die objektorientierte Konzepte wie Klassen, Vererbung und Polymorphismus einführt. Ähnlich der Z-Notation nutzt Vienna Development Method - Specification Language (VDM-SL) [69] Mengenlehre und Logik, legt aber zusätzlich einen stärkeren Fokus auf die operationelle Semantik. VDM-SL ist bekannt für seine Fähigkeit, sowohl abstrakte als auch konkrete Modelle zu beschreiben, was eine inkrementelle Spezifikation und Entwicklung von Systemen ermöglicht. Die Spezifikationssprache AlloyAlloy [70] unterscheidet sich von Z-Notation und VDM-SL durch die Verwendung einer relationalen Logik und bietet eine eigene Modellierungssprache, die für die Beschreibung von Strukturen und deren Beziehungen untereinander konzipiert ist. Unified Modeling Language (UML) [71] ist eine der bekanntesten modellbasierten Spezifikationssprachen. Sie bietet eine grafische Notation zur Darstellung verschiedener Aspekte eines Softwaresystems, wie z.B.: Struktur, Verhalten, Interaktionen und Implementierung. Der Entwicklungstrend modellbasierter Spezifikationssprachen zeichnet sich durch eine zunehmende syntaktische und semantische Annäherung an Implementierungssprachen aus. Nach [64] verfolgt die Angleichung zwei wesentliche Ziele: Erstens soll durch eine gesteigerte Vertrautheit die Akzeptanz in der Praxis erhöht werden. Zweitens zielt sie darauf ab, Synergien zwischen Spezifikation und Implementierung zu schaffen, um eine Entwicklung zu ermöglichen, die durch iteratives Verfeinern gekennzeichnet ist.

Übergangsbasierte Systeme stellen eine Modellierungsmethode für Systemverhalten dar, die charakterisiert ist durch ihre graphische Syntax und sich auf Zustandsübergänge konzentriert. Diese Klassen vom Methoden erfassen das Verhalten eines SuT als eine Menge von Zuständen und Übergänge als Stimuli, die das System zum Zustandswechsel veranlassen. Sie modellieren die Ausgabe eines Systems entweder als Eigenschaft der Zustände (wie bei der Moore-Maschine) oder der durchlaufenen Übergänge (wie bei der Mealy-Maschine).Modellbasiertes Testen [65] hat zur Anwendung von Übergangsbasierte Systeme in der Testmethodik beigetragen. Diese Methode verwendet Mo-

delle des SuT, um automatisiert Testfälle zu erzeugen. Diese Fälle simulieren und verifizieren systematisch das Verhalten des Systems unter variantenreichen Bedingungen. Indem das Systemverhalten mittels seiner Spezifikation in einem abstrakten Modell mit definierten Zuständen und Zustandsübergängen abgebildet wird, fördert modellbasiertes Testen eine effiziente und systematische Überprüfung der Systemfunktionalität. Es unterstützt dabei die Generierung von Testfällen, die umfassende Verhaltensbereiche des Systems abdecken.

Ein bekanntes Beispiel für übergangsbasierte Systeme stellen die endlichen Automaten, im speziellen Mealy- und Moore-Maschinen, dar. Mealy-Maschinen generieren ihre Ausgabe basierend auf dem aktuellen Zustand und der Eingabe, während Moore-Maschinen ihre Ausgabe ausschließlich vom aktuellen Zustand abhängig machen [72]. Ein STO, das auf diesen Prinzipien beruht, kann präzise vorhersagen, welche Ausgabe oder welcher Zustand von einem SuT erwartet wird. Bei Mealy-Maschine ermöglicht das Testorakel eine feingranulare Überprüfung der Systemreaktionen auf Eingaben in jedem Zustand, während bei Moore-Maschine das Testorakel sich auf die Überprüfung der zustandsabhängigen Ausgaben konzentriert. Simulink/Stateflow [73], eine Software für die Modellierung und Simulation von dynamischen Systemen, erweitert die Konzepte von Mealy- und Moore-Maschinen, indem es eine visuelle Umgebung für die Entwicklung von Steuerungs- und Signalverarbeitungssystemen bietet. Simulink/Stateflow findet Anwendung in der Entwicklung von Fahrerassistenzsystemen für Fahrzeugsimulationen, indem es zur Entwicklung und Simulation von Systemen wie Abstandstempomaten und Spurhalteassistenten verwendet wird. In diesem Kontext kann Simulink/Stateflow als STO eingesetzt werden, indem es ermöglicht, das erwartete Verhalten dieser Systeme zu definieren und zu simulieren.

3.1.2 Assertions und Verträge

Das Konzept der Assertions geht auf Alan Turing zurück [74]. Turing erkannte die Bedeutung einer Trennung zwischen Tester und Entwickler und empfahl, dass die Kommunikation zwischen ihnen über Assertions erfolgen sollte. Assertions sind boolsche Ausdrücke in der Implementierung, die an bestimmten Stellen in ein Programm eingefügt werden, um dessen Verhalten zur Laufzeit zu überprüfen. Sie dienen als interne Kontrollmechanismen, die während der Entwicklung und beim Testen helfen. Assertions prüfen Bedingungen wie die

Gültigkeit von Variablenwerten oder den Erreichungsstatus bestimmter Programmcodeabschnitte und zeigen eine Ausnahme, wenn eine Bedingung nicht erfüllt ist. Ihr primärer Zweck ist die Erkennung von Programmierfehlern durch Überprüfung von Invarianten innerhalb einer Methode oder Funktion.Ein typisches Beispiel für eine Assertion ist die Überprüfung der Gültigkeit eines Indexes in einem Array, bevor auf ein Element zugegriffen wird. Assertions sind in Programmiersprachen durch spezielle Konstrukte oder Standardbibliotheksfunktionen direkt unterstützt. So bieten C, C++ und Java ein Konstrukt namens Assert und C# eine Debug-Assert-Methode an. Die Einbettung von Assertions direkt in eine Implementierungssprache hat zwei wesentliche Auswirkungen, die sie von Spezifikationssprachen abgrenzen [64]. Erstens können Assertions direkt auf Programmvariablen verweisen und Beziehungen zwischen ihnen definieren. Dadurch wird die Diskrepanz zwischen Spezifikation und Implementierung hinsichtlich der Eigenschaften, die eine Assertion ausdrücken und überprüfen kann, verringert. In diesem Sinne sind Assertions eine natürliche Folge der Entwicklung von Spezifikationssprachen hin zur Unterstützung der Entwicklung durch iterative Verfeinerung. Zweitens werden Asserts zusammen mit dem Programmcode erstellt, dessen Laufzeitverhalten sie überprüfen, im Gegensatz zu Spezifikationssprachen, die der Implementierung vorausgehen.

Contracts [75] erweitern das Konzept der Assertions, indem sie nicht nur interne Zustände überprüfen, sondern auch die Beziehung zwischen verschiedenen Komponenten eines Systems definieren. Ein Contract für eine Methode oder Funktion spezifiziert formale, präzise und überprüfbare Anforderungen an die Eingaben (Preconditions), die Ausgaben (Postconditions) und die Invarianten, die während der Ausführung gelten müssen. Diese Verträge sind Teil der Schnittstellendefinition und dienen der Sicherstellung, dass die Interaktionen zwischen verschiedenen Teilen eines Programms den definierten Erwartungen entsprechen. Houssem et al. [76] nutzen Contracts als eine Methode zur Bewertung der Sicherheit von überwachten ML-Modellen, die kontinuierlich weiterlernen und per Over-The-Air (OTA) aktualisiert werden.

3.2 Abgeleitete Testorakel als Pseudoorakel

Abgeleitete Testorakel (ATO) stellen eine Kategorie von Testorakeln dar, die erwartete Ergebnisse aus dem System selbst ableiten, ohne auf vordefinier-

te Kriterien oder subjektive Urteile angewiesen zu sein. Im Gegensatz zu spezifischen Orakeln, die explizite Vorgaben nutzen, und Implizite Testorakel (ITO), die auf menschlicher Beurteilung basieren, erschließen ATO die Gültigkeit von Testergebnissen durch Analyse des Systemverhaltens, der Implementierung oder anhand anderer Artefakte. Diese Orakel sind besonders in Situationen von Bedeutung, in denen STO, die direkt über die Korrektheit einer Ausgabe urteilen können, entweder nicht verfügbar oder praktisch unanwendbar sind. Im Gegensatz zu STO, die auf einer vordefinierten Menge von Eingabe-Ausgabe-Paaren basieren oder eine formale Spezifikation des erwarteten Verhaltens nutzen, leiten ATO die Korrektheit der Software aus anderen Quellen ab. Diese Quellen können frühere Versionen der Software, äquivalente Systeme, frühere Testergebnisse oder Annahmen über das Verhalten der Software sein.

Ein frühes Konzept in der Entwicklung ATO ist die Einführung des Pseudo-Orakels durch Davis und Weyuker [77], entwickelt für den Umgang mit sogenannten nicht-testbaren Programmen. Diese Programme werden definiert als:

> "Programme, die erstellt wurden, um eine Antwort zu ermitteln, die vorab nicht bekannt ist. Es besteht kein Anlass, derartige Programme zu verfassen, sollte die korrekte Antwort bereits feststehen. [77]"

Ein Pseudo-Orakel repräsentiert eine alternative Umsetzung des zu prüfenden Systems, die eigenständig von einem anderen Entwicklerteam oder in einer anderen Programmiersprache entwickelt wurde. Da keine Gewissheit über die Fehlerfreiheit eines Orakels existiert, erhält das Pseudo-Orakel seinen Namen. Bei Nichtübereinstimmung der Ausgaben von SuT und Orakel ist ein Debugging notwendig, um den Fehlerort zu identifizieren. Dieses Prinzip findet auch Anwendung in der fehlertoleranten Datenverarbeitung, insbesondere bei der sogenannten Mehrfach- oder N-Versionen-Programmierung (NVP) [78]. Die NVP ist eine bei der N verschiedene Implementierungen desselben Algorithmus erstellt und gleichzeitig ausgeführt werden. Die Ausgaben der verschiedenen Implementierungen werden mittels eines Abstimmungsmechanismus verglichen, um das valide Ergebnis zu bestimmen. Auf diese Weise dienen die divergierenden Implementierungen als abgeleitete Testorakel zur Überprüfung der Systemzuverlässigkeit. Das Paradigma der Erstellung multipler, funktional äquivalenter Softwareimplementierungen, ist von zentraler Bedeutung für

Anwendungen mit hohen Anforderungen an Fehlertoleranz, insbesondere in der Luftfahrttechnologie. In [79] wird dokumentiert, wie die Anwendung von NVP das Risiko korrelierter Fehler reduziert, indem sie die Entwicklung eines automatischen Flugzeuglandungsprogramms auf 15 unabhängige Teams verteilt.

3.3 Implizierte Testorakel

ITO bewerten die Richtigkeit von Testausgaben, ohne dass spezifisches Domänenwissen oder formale Spezifikationen notwendig sind, was sie für eine breite Palette ausführbarer Programme nutzbar macht. Sie finden primär Anwendung in der Identifikation von Anomalien, wie Programmabbrüchen, Deadlocks oder Livelocks, die als eindeutige Fehler angesehen werden können, selbst ohne tiefergehende Informationen [80]. Ein Fehlverhalten, das spezifisches Verhalten betrifft, kann durch diese Art der Testorakel nicht erkannt werden. Des Weiteren hängt die Effektivität ITO vom spezifischen Kontext ab; eine universelle Anwendbarkeit ITO auf sämtliche Programmarten existiert nicht. Eine Verhaltensweise, die in einem gegebenen Kontext als fehlerhaft klassifiziert wird, kann in einem differierenden Kontext oder innerhalb eines anderen Systems als adäquates Verhalten bewertet werden.Die wissenschaftliche Auseinandersetzung mit ITO lässt sich in vier Hauptforschungsbereiche gliedern:

- Analyse von Problemen der Systemkonkurrenz
- Evaluation nicht-funktionaler Eigenschaften wie Performanz, Robustheit und Speichermanagement
- Fuzzing, definiert durch die Erzeugung zufälliger Eingabedaten zur Identifikation von Anomalien
- Identifikation von Anomalien in der Funktionsweise von Webapplikationen

Fuzzing stellt eine Methode zur Identifikation impliziter Anomalien dar [81]. Dieser Ansatz basiert auf der Generierung zufälliger oder semi-zufälliger Daten, den sogenannten „Fuzz“-Eingaben, mit denen ein System „angegriffen“ wird, um potenzielle Schwachstellen oder Fehlverhalten aufzudecken. Der Mechanismus des Fuzzings liegt in seiner Fähigkeit, durch diese zufällige

Eingaben unvorhergesehene Systemreaktionen zu provozieren, die unter regulären Testbedingungen unentdeckt bleiben würden. Im Falle der Erkennung einer Anomalie, sei es durch unerwartetes Verhalten oder direkte Systemfehler, dokumentiert der Fuzz-Testprozess präzise die spezifischen Eingaben oder Eingabesequenzen, die zu dem jeweiligen Fehlverhalten geführt haben. Die praktische Anwendung von Fuzzing-Techniken ist insbesondere im Bereich der Cybersicherheit von Bedeutung. Hierbei wird Fuzzing eingesetzt, um Sicherheitslücken aufzudecken, darunter Buffer Overflows, Speicherlecks, unbehandelte Ausnahmen und Denial-of-Service-Angriffe [82].

3.4 Bewertung der Testorakeltypen und Diskussion

Die Eigenschaften der betrachteten Testorakel werden in Abbildung 3.2 zusammengefasst und allgemein, ohne Betrachtung des SuT anhand der Präzision, Aufwand, Adaptivität, Transparenz, und Systemwissen bewertet.

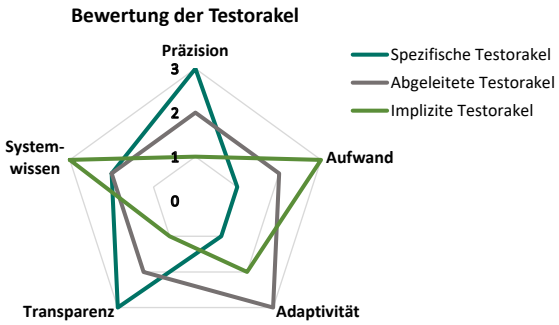

Abbildung 3.2: Vergleich spezifischer, abgeleiteter und impliziter Testorakel anhand ausgewählter Kriterien.

Präzision: Dieses Kriterium bezieht sich darauf, wie genau ein Testorakel feststellen kann, ob das getestete System das erwartete Verhalten zeigt oder nicht. Spezifische Testorakel bieten die höchste Genauigkeit und sind damit am zuverlässigsten, wenn es um die exakte Überprüfung des Systemverhaltens geht. Implizite Testorakel weisen die geringste Genauigkeit, da sie auf

allgemeinen Annahmen basieren und keine tiefergehenden Überprüfungen ermöglichen.

Aufwand: Der Aufwand zur Implementierung und Nutzung eines Testorakels wird determiniert durch die Gesamtheit der benötigten Ressourcen. Dies umfasst zeitliche Investitionen, finanzielle Mittel, materielle und immaterielle Ressourcen. Der Aufwand für die Implementierung ist bei spezifischen und impliziten Testorakeln hoch, hauptsächlich wegen der Spezifikationsanforderungen. Abgeleitete Testorakel sind hier vorteilhafter, da sie weniger Aufwand für die Einrichtung und Anwendung

Adaptivität: Adaptivität bezieht sich auf das Vermögen eines Testorakels, seine Funktionsweise flexibel und zeitnah an veränderte Anforderungen, Spezifikationen und Systemumgebungen anzupassen. Ein adaptives Testorakel zeigt Robustheit gegenüber Veränderungen in Softwareversionen, Konfigurationen und operativen Umgebungen. Abgeleitete und implizite Testorakel bieten eine hohe Anpassungsfähigkeit und eignen sich für dynamische und sich verändernde Systemumgebungen. Spezifische Testorakel sind in dieser Hinsicht limitiert, da jede Änderung im System eine Überarbeitung der Spezifikationen erfordert.

Transparenz: Transparenz innerhalb des Kontexts von Testorakeln adressiert das Maß, in dem die Entscheidungsfindung und Ergebnisermittlung für den Benutzer nachvollziehbar und verständlich sind. Ein transparentes Testorakel ermöglicht es, die Ableitung seiner Urteile zu verstehen, und fördert somit Vertrauen und Akzeptanz. Spezifische Testorakel sind hochtransparent, da sie auf definierten Anforderungen und Spezifikationen basieren. Implizite Testorakel könnten als nicht transparent eingestuft werden, besonders wenn sie nur in der Lage sind, Systemabstürze zu erkennen, ohne Aufschluss über die zugrunde liegende Fehlerursache zu geben.

Systemwissen: Dieses Kriterium evaluiert den Umfang und die Tiefe des Wissens über das zu testende System, das für die Funktionsweise des Testorakels erforderlich ist. Systemwissen kann sich auf die Kenntnis der internen Struktur, externen Schnittstellen, der dynamischen Verhaltensweisen sowie der Systemumgebung beziehen. Die Nutzung spezifischer Testorakel verlangt ein hohes Maß an Fachwissen, um Spezifikationen zu entwickeln und zu interpretieren. Implizite Testorakel hingegen kommen mit dem geringsten Bedarf an spezialisiertem Wissen aus, da sie auf generellen Kriterien wie der Systemstabilität basieren.

Die Bewertung der verschiedenen Arten von Testorakeln basiert auf den definierten Anforderungen (siehe Kapitel 1.2), sowie auf den darauffolgenden Diskussionen hinsichtlich ihrer Eignung für SSIB (nach Gleichung 1.8).

(a) Benutzeraktionen und die Grenzen spezifischer Testorakel: Ein zentrales Problem beim Testen selbstlernender Systeme stellt die Schwierigkeit dar, Benutzeraktionen vollumfänglich zu definieren. Diese Unbestimmtheit erschwert den Einsatz STO, die eine Antizipation des erwarteten Verhaltens voraussetzen. Alternativ bieten sich abgeleitete und implizite Testorakel an, die ohne Spezifikationen auskommen und somit die erforderliche Flexibilität im Testprozess gewährleisten. **(b) Selbstlernende Systeme und die Notwendigkeit plausibler Verhaltensprüfungen:** Im Kontext selbstlernender Systeme ist die Fähigkeit eines Testorakels, die Plausibilität von erlerntem Benutzerverhalten adäquat zu bewerten, von zentraler Bedeutung. ITO, ausgerichtet auf die Identifikation systemkritischer Fehler wie Abstürze, bieten keine Möglichkeit für die Evaluierungen. Im Gegensatz dazu erlauben abgeleitete und spezifische Testorakel eine Analyse, die über die reine Fehlererkennung hinaus die Eignung des erlernten Verhaltens kritisch hinterfragt. Diese Betrachtungsweise ist insbesondere für das in dieser Dissertation verfolgte Ziel, plausible Benutzeraktionen zu erlernen und zu validieren, von Relevanz. **(c) Bedeutung der Anpassungsfähigkeit:** Die Adaptivität selbstlernender Systeme ist ein zentrales Element, das es ihnen ermöglicht, sich im Laufe der Zeit zu entwickeln. ATO können diese dynamischen Veränderungen berücksichtigen. Sie stützen sich auf flexible Informationsquellen wie Ergebnisse früherer Tests, um Anpassungen des Systems zu erfassen. Diese Anpassungsfähigkeit erlaubt eine kontinuierliche Validierung des Systemverhaltens, selbst bei dem Entstehen neuer Benutzeraktionen. Somit stellen abgeleitete Testorakel eine Gruppe von Methoden dar, die sowohl die Funktionalität als auch die Lernfähigkeit des Systems im Zeitverlauf sicherstellt.

Zusammenfassend wird festgestellt, dass unter den Kategorien von Testorakeln die Gruppe, der ATO geeignet ist für die Validierung von selbstlernenden Systemen mit Benutzeraktionen. Ihre inhärente Fähigkeit zur Anpassung an die Dynamik selbstlernender Systeme, kombiniert mit der Fähigkeit, ein Spektrum an Benutzeraktionen abzudecken, macht sie zu einem Instrument für eine effektive Testumgebung.

4 Methoden der Testfallerstellung für abgeleitete Testorakel

Im Anschluss an die in Abschnitt 3.4 durchgeführte Diskussion erweisen sich ATO als geeignete Variante von Testorakeln für die vorgesehenen Anwendungen, da insbesondere die Kategorie „Adapativität“ für die Validierung von Benutzeraktionen von Relevanz ist. Das Konzept der abgeleiteten Testorakel (siehe Abschnitt 3.2) beschrieben, bildet die Grundlage für Methoden, die zur Realisierung abgeleiteter Testorakel eingesetzt werden können.

4.1 Kreuzreferenzierung als abgeleitetes Testorakel

Die Kreuzreferenzierung, angewendet als ATO, ermöglicht die Validierung von Softwareoutputs durch den Vergleich mit den Ergebnissen alternativer, als korrekt betrachteter Implementierungen. Diese Methode beruht auf der Annahme, dass für ein gegebenes Problem verschiedene Lösungsansätze unter korrekter Ausführung konsistente Ergebnisse liefern sollten [64]. Der Vergleich dieser Outputs dient der Identifikation von Inkonsistenzen, welche potenzielle Fehlerquellen in der getesteten Software aufzeigen können. Diese Vorgehensweise erweist sich insbesondere in Situationen als wertvoll, in denen spezifische Output-Erwartungen entweder nicht definiert oder nicht zu prognostizieren sind. Nach [64] lassen sich zwei Hauptmethoden des Vergleichs unterscheiden:

- **Vergleich von Testergebnissen:** Die Ergebnisse unterschiedlicher Implementierungen oder Versionen desselben Programms werden gegenübergestellt, um Differenzen zu untersuchen (siehe Unterabschnitt 4.1.1).

- **Vergleich mit Referenzsystemen:** Die Software wird mit einem bestehenden Referenzsystem verglichen, das aufgrund seiner Validität als verlässlich gilt (siehe Unterabschnitt 4.1.2).

4.1.1 Differenzielles Testen

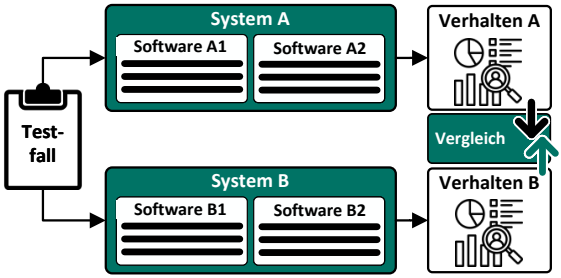

Abbildung 4.1: Schematische Darstellung des differenziellen Testverfahrens, bei dem identische Testinputs gleichzeitig in zwei verschiedene Systeme eingegeben werden. Die daraus resultierenden Verhaltensweisen werden nachfolgend verglichen, um Abweichungen zu ermitteln und die Korrektheit der Systeme zu überprüfen.

Differenzielles Testen ist eine Testmethode, die durch den Vergleich der Verhalten von zwei oder mehr Versionen eines Softwaresystems derselben Spezifikation auf Unstimmigkeiten in deren Verhalten abzielt (siehe Abbildung 4.1). Diese Methode gründet auf der Prämisse, dass Implementierungen, die konzeptionell identische Funktionen ausführen, bei gleichen Eingaben entsprechend identische Ausgaben generieren sollten. [77, 83]

Die Methode dient der Fehlererkennung, speziell bei Compilern [84], ML-Modellen und Deep-Learning-Bibliotheken (siehe Abbildung 2.1), indem sie Abweichungen zwischen dem antizipierten und dem realen Verhalten unterschiedlicher Systeme offenlegt. Die Forschung von Nejadgholi und Yang [85] zeigen, dass 5% bis 27% der Testorakel für Deep-Learning-Bibliotheken Differenzielles Tests anwenden.

Srisakaokul et al. [86] haben das Konzept des Differenziellen Testen für das Testen ML-Modellen vorgestellt, mit einem Fokus auf Algorithmen des überwachten Lernens.

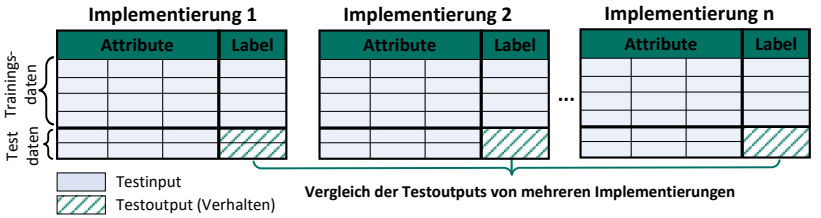

Abbildung 4.2: Schematische Darstellung des differenziellen Testansatzes, bei dem ein Testinput auf mehrere Implementierungen angewendet wird. Jede Implementierung führt Trainings mit identischen Attributen und Labels durch, woraufhin die resultierenden Testoutputs zum Aufdecken von Inkonsistenzen untereinander verglichen werden (nach [86])

Dieser Ansatz verwendet n unterschiedliche Implementierungen desselben ML-Modells, wobei speziell die erste Implementierung, als das SuT betrachtet wird. Die Methodik wird anhand von $n = 3$ Testfällen mit identischen Trainings- und Testdatensätzen demonstriert (siehe Abbildung 4.2). Das als korrekt angesehene Label für jede Implementierung ergibt sich aus der Mehrheit der Labels (mehr als 50%) aus den Testoutputs der n Implementierungen. Sollte das Label der ersten Implementierung von diesem Mehrheitslabel abweichen, deutet dies auf einen Fehler in dieser spezifischen Implementierung hin. Das Evaluieren des Konzeptes erfolgte anhand des Testoutputs zweier maschineller Lernalgorithmen (k-Nearest Neighbor (k-NN) und Naive-Bayes-Algorithmus (NB)). Es konnten 16 Fehler in 7 Implementierungen des NB sowie 13 Fehler in 19 Implementierungen des k-NN identifiziert werden, ohne dass es erforderlich war, ein spezifisches erwartetes Verhalten festzulegen, wie es bei einem STO notwendig wäre.

DeepXplore [87] nutzt ein differenzielles System zum systematischen Testen von Systemen, basierend auf tiefen neuronalen Netzen sowie zur automatischen Erkennung von fehlerhaftem Verhalten, ohne die Notwendigkeit von manueller Datenbeschriftung. Mehrere DL-Systeme werden mit ähnlichen Funktionen als Referenzorakel eingesetzt, um eine händische Überprüfung zu vermeiden. Eines der DL-Systeme dient dabei als Referenz für die verbleibenden Systeme. In Anlehnung an die Testabdeckung bei traditionellen Softwaretests, verwenden die Autoren den Grad der Neuronenabdeckung, um die Testgenerierung zu leiten. Zusätzlich werden synthetische Testdaten generiert, die voneinander

unterschiedliche Testergebnisse liefern und dabei gleichzeitig so realistisch wie möglich sind.

Im Kontrast zu den Veröffentlichungen von Pei et al. [87] und Srisakaokul et al. [86], welche differenzielles Testen zur Identifizierung von Fehlern in ML-Modellen einsetzen, fokussieren Pham et al. [88] auf die Nutzung dieser Methode zur Aufdeckung von Defekten in Deep-Learning-Bibliotheken, die für die Entwicklung von ML-Modellen verwendet werden. Der von [88] vorgeschlagene differenzielle Testansatz CRADLE[1] implementiert:

1. Eine konsistenzbasierte Überprüfung über verschiedene Implementierungen hinweg zur Identifikation von Fehlern in Deep-Learning-Bibliotheken.
2. Die Nachverfolgung und Analyse der Ausbreitung von Anomalien, um die fehlerhaften Komponenten innerhalb dieser Bibliotheken zu lokalisieren.

Beim Testen von DL-Bibliotheken kann die erwartete Ausgabe für eine spezifische Eingabe oftmals nicht bekannt sein. Allerdings können Differenzielle Tests mit unterschiedlichen Implementierungen desselben Algorithmus aus verschiedenen DL-Bibliotheken miteinander verglichen werden. Die Testmethode CRADLE wurde anhand von drei DL-Bibliotheken – TensorFlow, CNTK und Theano – sowie 11 Datensätzen und 30 vortrainierten Modellen evaluiert. Diese Evaluation identifizierte 12 Fehler und 104 inkonsistente Verhaltensweisen durch CRADLE und konnte darüber hinaus die spezifischen Funktionen identifizieren, die mit den festgestellten Inkonsistenzen assoziiert sind.

Wang et al. [89] erweiterten den Ansatz von CRADLE mit der Entwicklung von LEMON[2], einem Testverfahren für DL-Bibliotheken mittels gesteuerter Mutation. LEMON nutzt zwölf Mutationsregeln, um DL-Modelle zu verändern und dadurch Aufrufsequenzen in Bibliothekscodes sowie schwer zu detektierende Verhaltensweisen zu generieren. Zusätzlich implementiert LEMON eine heuristische Strategie zur Navigation des Modellgenerierungsprozesses, die darauf abzielt, Modelle zu erzeugen, welche den Grad der Inkonsistenz zwischen verschiedenen Bibliotheken maximieren. Diese Methode hat ihre

[1] Cross-Backend Validation to Detect and Localize Bugs in Deep Learning Libraries
[2] deep learning Library tEsting via guided MutatiON

Effektivität unter Beweis gestellt, indem LEMON 24 bisher unbekannte Fehler in den neuesten Versionen von TensorFlow, Theano, CNTK und MXNet identifizieren konnte.

4.1.2 Regressionstests

Regressionstests (siehe Abbildung 4.3) zielen darauf ab, nach Modifikationen des Programmcodes die Integrität bestehender Funktionen zu gewährleisten. In Situationen, in denen ATO eingesetzt werden und explizite Testoutputspezifikationen nicht vorhanden oder unvollständig sind, basieren Regressionstests auf dem Vergleich der aktuellen Softwareausgaben mit den früheren, validierten Outputs. Diese Methode geht von der Annahme aus, dass vorherige Systemversionen als zuverlässige Testorakel für die Validierung bestehender Funktionen dienen. [90]

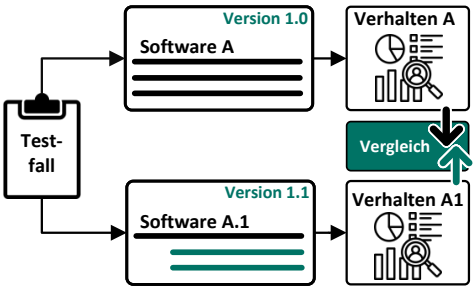

Abbildung 4.3: Schematische Darstellung des Regressionstests, bei dem zwei Versionen eines Programms miteinander verglichen werden.

Nach [64] wird zwischen zwei Arten von Programmänderungen unterschieden:

- Korrektive Änderungen: Diese zielen darauf ab, identifizierte Fehler zu korrigieren. Testfälle der vorherigen Version, die als Testorakel fungieren, sind auch für die überarbeitete Version anwendbar.
- Perfektionierende Änderungen: Diese erweitern das Programm um neue Funktionalitäten, was eine Aktualisierung der vorhandenen Testfälle erfordert, um die neuen Features abzudecken. Diese Aktualisierungen be-

inhalten Anpassungen sowohl der Testdaten als auch der erwarteten Ergebnisse, was eine Modifikation des Testorakels nach sich zieht.

Bei korrektiven Änderungen, die darauf ausgerichtet sind, ursprünglich beabsichtigte Funktionen zu erhalten oder wiederherzustellen, kann die frühere Version als verlässliches Testorakel für die überarbeitete Version dienen. Es besteht jedoch die Gefahr, dass solche Änderungen das zugrundeliegende Problem nicht adäquat adressieren oder bestehende Funktionen negativ beeinflussen. In diesen Fällen können Testorakel effektiv durch symbolischer Vergleich - die Analyse der Ausführungsunterschiede zwischen der defekten und der mutmaßlich korrigierten Version – erstellt werden, wie von Gu et al. [91] vorgeschlagen.

Xie et al. [92] entwickelten einen automatisierten Ansatz namens Orstra, der es ermöglicht, automatisch erstellte Testfälle mittels Regressionstests zu validieren. Orstra erfasst Objektzustände während der Softwareausführung und leitet daraus Assertions ab. Bei Modifikationen der Software können durch Orstra Assertion-Verletzungen aufgedeckt werden. Orstra wurde erfolgreich an automatisch generierten Testfällen für elf verschiedene Softwarekomponenten aus verschiedenen Quellen erprobt. Die Ergebnisse bestätigen, dass Orstra die Fähigkeit automatisch generierter Tests zur Fehlererkennung durch Bereitstellung von Regressionsorakeln steigern kann. Jedoch ist Orstra auf die Überprüfung korrektiver Änderungen beschränkt; für perfektionierende Änderungen liegen zum Zeitpunkt dieser Dissertation keine wissenschaftlichen Studien vor, die eine Validierung durch Regressionstests dokumentieren.

4.2 Einsatz von Spezifikationsmining für abgeleitete Testorakel

Spezifikationsmining, als Methode zur Erstellung eines ATO, ist ein systematischer Ansatz zur automatischen Extraktion von Spezifikationen aus bestehenden Softwaresystemen. Die extrahierten Spezifikationen werden in Form von Invarianten dargestellt, die an bestimmten Programmstellen eingehalten werden müssen.

Definition 4.2 Invarianten: In der Softwareentwicklung bezeichnet eine Invariante eine Bedingung oder Regel, die im Verlauf der Programmausführung oder innerhalb eines bestimmten Abschnitts eines Softwareprozesses konstant gültig bleibt.

Diese extrahierten Spezifikationen können im Weiteren als Grundlage für STO verwendet werden (siehe Abschnitt 3.1). Die Ansätze des Spezifikationsmining unterscheiden sich in ihrer Methodik zur Extraktion von Invarianten aus dem Quellcode, wobei sie sich grundlegend in statische und dynamische Analyseverfahren aufteilen.

4.2.1 Statische Analyse

Bei der statischen Analyse werden Invarianten durch die Untersuchung des Quellcodes eines Programms ohne dessen Ausführung identifiziert. Dieser Ansatz analysiert den Code, um potenzielle Invarianten basierend auf der Struktur des Programms und den möglichen Pfaden durch den Code zu finden. Statische Analysewerkzeuge können Invarianten erkennen, die in allen möglichen Ausführungen des Programms gelten, können aber durch die Präsenz von Schleifen und rekursiven Aufrufen eingeschränkt sein.

Symbolische Ausführung

Bei der symbolischen Ausführung von Programmen werden anstatt konkreter Eingabewerte symbolische Werte verwendet werden, die alle möglichen Eingaben repräsentieren. Durch die Verwendung symbolischer Werte kann die symbolische Ausführung den Zustandsraum eines Programms systematisch erforschen und dabei verschiedene Pfade durch das Programm simulieren.

In [93] wird ein symbolischer Algorithmus eingeführt, der auf Binary Decision Diagrams (BDD) basiert und die vorhandene Regelmäßigkeit in der Darstellung möglicher Kombinationen von Systemkomponenten ausnutzt, um Spezifikationen zu finden. Diese Spezifikationen haben sich als nützlich erwiesen, um unbekannte Fehler in realen Systemen zu entdecken. Jaffar et al. [94] präsentierten eine erweiterte symbolische Ausführung mit Interpolation, um ungebundene Schleifen im Kontext der Programmspezifikation anzugehen. Der Zweck dieser Interpolanten in diesem Kontext ist es, zu demonstrieren, dass

bestimmte Fehlerzustände oder „Fehlerknoten“ im Programmcode unter keinen Umständen erreicht werden können. Ein Fehlerknoten ist ein Punkt im Programm, an dem ein potenzieller Fehler auftritt – beispielsweise ein Zustand, der zu einem Programmabsturz oder unerwünschtem Verhalten führt. Sie schlugen einen Algorithmus vor, der invariante Interpolanten entdeckt, die die Unreichbarkeit von Fehlerknoten beweisen.

4.2.2 Dynamische Analyse

Die dynamische Analyse zur Erkennung von Invarianten beinhaltet die Ausführung des Programms mit verschiedenen Eingabedatensätzen und die Beobachtung des Verhaltens zur Laufzeit, um Invarianten zu identifizieren. Oftmals ziehen diese Methoden Nutzen aus Datenanalyseverfahren oder Techniken des maschinellen Lernens.

Datenanalyseverfahren

In den letzten Jahren hat die Forschung zur Verwendung von dynamischen Spezifikationsmining zur Modellierung des Verhaltens von Softwaresystemen für Testorakel zugenommen [95–97] Diese Techniken durchlaufen einen dreistufigen Prozess:

1. Datentransformation: Programmausführungsspuren werden in alternative Darstellungen umgewandelt, beispielsweise Itemsets (zitiert in [95]), Graphen (zitiert in [97]) oder andere Formate.

2. Mustererkennung: Data-Mining-Algorithmen werden auf die transformierten Daten angewendet, um wiederkehrende Muster aufzudecken. Beispiele hierfür sind Itemsets oder Subgraphstrukturen. Diese Muster werden verwendet, um Programmierregeln abzuleiten.

3. Erkennung von Fehlern: Der letzte Schritt umfasst die Identifizierung von Verstößen gegen die festgelegten Regeln.

Ernst et al. [98] stellen Daikon vor, eine Methode zum automatischen Ableiten von Dateninvarianten durch dynamische Analyse. Daikon bezieht Programmvariablen aus dynamischen Ausführungsspuren ein und lernt wahrscheinliche Invarianten bezüglich Datenwertbeziehungen. Daikon ist eine Softwareanwen-

dung, die mit einer Reihe von Testfällen und einer Reihe von potenziellen Invarianten arbeitet. Diese Invarianten werden aktiviert, indem ihre Variablen mit den Variablen des Programms verknüpft werden. Daikon leitet dann wahrscheinliche Invarianten aus denjenigen ab, die während der Ausführung des Programms auf den Eingaben nicht verletzt wurden. Diese abgeleiteten Invarianten spiegeln das Verhalten des Programms wider und können zur Überprüfung seiner Genauigkeit herangezogen werden. Bei Regressionstests können beispielsweise aus der vorherigen Version abgeleitete Invarianten verifiziert werden, um ihre Gültigkeit in der aktualisierten Version zu bestimmen.

Csallner et al. [99] haben in ihrem Tool namens DySy die Vorteile der dynamischen Invarianteninferenz mit Daikon und der statischen Analyse mit symbolischer Ausführung (Unterunterabschnitt 4.2.1) kombiniert. Dadurch gelingt es, eine genauere Menge an Invarianten abzuleiten als mit Daikon allein.

ML-basierte Testorakel

ML wird in den vergangenen Jahren zunehmend eingesetzt, um das Testorakelproblem in der Softwareentwicklung zu adressieren. Ein Schlüsselindikator für die Effektivität eines ML-basierten Testorakels ist die Genauigkeit (siehe 2.7), definiert als der Anteil der Eingaben, die vom Modell korrekt klassifiziert werden. Empirische Studien [100] zeigen, dass Techniken wie KNN, Support Vector Machine (SVM), k-NN und NB in der Lage sind, eine Genauigkeit von über 90% zu erzielen. KNN sind die am weitesten verbreiteten Mustererkennungsalgorithmen im ML und werden zunehmend als Testorakel eingesetzt.

Vanmali et al. [101] entwickelten ein automatisiertes Testorakel, basierend auf KNN, zur Simulation des Verhaltens einer Software anhand ihrer vorherigen Version. Dieses Modell wurde speziell für den Einsatz in Regressionstests entworfen. Aggarwal et al. [102] nutzten ein trainiertes KNN zur Validierung von Klassifizierungsproblemen. Shahmiri et al. [103] setzten KNN-basierte Testorakel ein, um Entscheidungsstrukturen zu prüfen und logische Module zu validieren.

Im Gegensatz zu den vorherigen Ansätzen, die ein einzelnes KNN für die Erzeugung der erwarteten Ausgaben verwendeten, präsentiert Shahamiri et al. [104] ein Testorakel, das mehrere KNNs nutzt, wobei jedes für eine spezifische erwartete Ausgabe verantwortlich ist. Diese Methode erlaubt es, in Softwaresystemen, die Szenarien benötigen, Fehler präziser zu identifizieren.

Ein Nachteil dieser Methode ist jedoch, dass das Training mehreren KNNs mehr Zeit und Speicherplatz beansprucht als das eines einzelnen KNN.

Deep Billboard [105] generiert Kamerabilder von Werbetafeln am Straßenrand, die verwendet werden, um das korrekte Verhalten von tiefen künstlichen neuronalen Netzen für das autonome Fahren zu testen. Das Konzept wurde anhand des DARPA Autonomous Vehicle (DAVE) [106] validiert. DAVE ist ein Fahrzeug, ausgestattet mit zwei Kameras, das seine autonome Fahrfunktion durch Ende-zu-Ende-Lernen unter Einsatz von CNNs erworben hat. Ende-zu-Ende-Lernen ist eine Lernmethode, die Lenkanweisungen direkt aus Kamerabildern erzeugt. Der Test mit Deep Billboard zeigt, dass bei einem Kamerabild mit leerem Billboard ein korrektes Lenkverhalten vorliegt (siehe Abbildung 4.4, links), wohingegen das Hinzufügen eines gegnerischen Billboards zu einem falschen Verhalten führt (siehe Abbildung 4.4, rechts).

Abbildung 4.4: Das künstliche Hinzufügen einer Werbetafel am Straßenrand (rechts) resultiert in einem modifizierten Lenkwinkel im Vergleich zur Fahrbahn ohne Werbetafel (links). [105]

4.3 Fortschritte und Anwendungsbereiche des metamorphen Testens

Metamorphes Testen (MT) stellt eine Teststrategie dar, die auf dem Konzept der Metamorphen Beziehungen (MB)[3] basiert. Das Verfahren wurde erstmals 1998 von Chen et al. [107] vorgeschlagen und war ursprünglich nur als Methode

[3] Das Wort „metamorph“ leitet sich aus dem Griechischen ab: der Wortteil „meta“ bedeutet Veränderung, der Wortteil „morph“ steht für Form oder Gestalt.

zur Erzeugung erfolgreicher Testfälle durch Transformation konzipiert, nicht als eine Technik zur Anwendung als Testorakel.

MT ist eine Methode zur Softwarevalidierung, die eingesetzt wird, wenn für einzelne Testfälle keine bekannten Ergebnisse vorliegen. Anstatt die Korrektheit eines einzelnen Outputs zu überprüfen, wird der Zusammenhang zwischen einem Ausgangstestfall und einem Folgetestfall untersucht. Dabei wird zunächst ein Ausgangstestfall definiert, für den die Software ein Ergebnis liefert, das jedoch nicht als richtig oder falsch eingeschätzt werden kann, weil kein erwartetes Ergebnis bekannt ist. Aus diesem Ausgangstestfall wird ein Folgetestfall abgeleitet, indem die Eingaben auf eine bestimmte Weise verändert werden (siehe Abbildung 4.5).

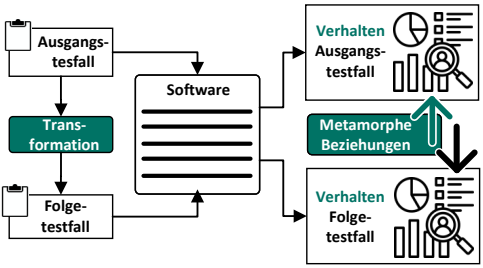

Abbildung 4.5: Darstellung von Metamorphen Testen und von Metamorphen Beziehungen zwischen dem Verhalten des Ausgangstestfalls und des Folgetestfalls

Zwischen dem Ausgangstestfall und dem Folgetestfall wird eine sogenannte MB (siehe Definition 4.4) erwartet, die festlegt, wie sich die Ergebnisse im Verhältnis zueinander ändern sollten. Diese Beziehung beschreibt, wie das Ergebnis des Folgetestfalls im Vergleich zum Ausgangsergebnis aussehen müsste, basierend auf der Transformation, die auf die Eingabedaten angewendet wurde. Das entscheidende Kriterium für die Validierung ist daher nicht das Ergebnis eines einzelnen Testfalls, sondern ob das Verhältnis zwischen den Ergebnissen von Ausgangs- und Folgetestfall den Erwartungen entspricht. Wenn dieses Verhältnis unerwartet abweicht, weist dies auf mögliche Fehler in der Software hin.

Im formalen Kontext können MB nach [108] wie folgt definiert werden:

Definition 4.4 Metamorphe Beziehungen: Sei f eine Ziel-Funktion oder ein Algorithmus. Eine metamorphe Beziehung (MB) ist eine notwendige Eigenschaft[a] von f über einer Sequenz von zwei oder mehr Eingaben $(x_1, x_2, \ldots, x_n)$, wobei $n \geq 2$, und deren entsprechenden Ausgaben $(f(x_1), f(x_2), \ldots, f(x_n))$. Sie kann als eine Beziehung $R \subseteq X^n \times Y^n$ ausgedrückt werden, wobei C die Teilmengenrelation darstellt und X^n und Y^n die kartesischen Produkte der Eingabe- und Ausgaberäume sind. Vereinfacht kann eine eine MB wie folgt dargestellt werden $B(x_1, x_2, \ldots, x_n, f(x_1), f(x_2), \ldots, f(x_n))$.

[a] Eine notwendige Eigenschaft eines Algorithmus bedeutet eine Bedingung, die logisch aus dem Algorithmus abgeleitet werden kann.

Bei der Verwendung dieser Methode wird das SuT als Blackbox betrachtet (siehe Abbildung 2.7), das heißt, es wird keine Veränderung des Programmcodes vorgenommen, im Gegensatz zum differenziellen Testen (siehe Abbildung 4.1). Die Implementierung von MB erfordert zunächst die Identifizierung MB für das untersuchte System, die domänenspezifisch sind und ein tiefgreifendes Verständnis des Problembereichs sowie der Funktionalitäten des Systems voraussetzen. Nach der Identifikation erlauben diese Beziehungen die Ableitung zusätzlicher Testfälle: Von einem Ausgangstestfall (Englisch: source test case) ausgehend, werden durch die Anwendung der MB ein oder mehrere Folgetestfälle (Englisch: follow-up testcase) generiert, deren Ausgaben in einer vorhersagbaren Beziehung zum Ergebnis des initialen Testfall stehen sollten (siehe Definition 4.6).

Definition 4.6 Ausgangs- und Folgetestfall: Gegeben sei eine MB $B(x_1, x_2, \ldots, x_n, f(x_1), f(x_2), \ldots, f(x_n))$. Für jedes $i = 1, 2, \ldots, k$ wird x_i als Ausgangstestfall bezeichnet. Für jedes $j = k+1, k+2, \ldots, n$ wird x_j als Folgetestfall bezeichnet. Somit kann wenn für eine gegebene Metamorphe Beziehung B, alle Ausgangstestfälle x_i (für $i = 1, 2, \ldots, k$) spezifiziert sind, die Folgetestfälle x_j (für $j = k+1, k+2, \ldots, n$) basierend auf den Ausgangstestfällen und, falls notwendig, auf deren entsprechenden Ausgaben konstruiert werden.

MT nutzt MB zur effizienten Erstellung neuer Testinstanzen durch gezielte Transformationen existierender Testfälle, was eine manuelle Generierung individueller Testfälle für jede Eingabe nicht notwendig macht.

Wurde bei der Anwendung des MT keine Verletzung MB festgestellt, kann lediglich bestätigt werden, dass der ML-Algorithmus nicht grundsätzlich falsch ist. Dies ist jedoch kein Beweis für die Korrektheit des ML-Algorithmus. Erst die Verletzung der MB lässt den Schluss zu, dass mindestens einer der beiden Testausgänge (Verhalten) falsch sind und somit eine fehlerhafte Implementierung des ML-Algorithmus vorliegt oder er für die gewählte Anwendung nicht geeignet ist [109].

4.3.1 Empirische Untersuchungen und Weiterentwicklungen des metamorphen Testens

Einer Untersuchung aus dem Jahr 2019 zufolge wurde MT anhand verschiedener Fallstudien zwischen 1998 und 2018 in insgesamt 84 verschiedenen Domänen untersucht [110]. Der größte Anteil wurde im Bereich Webdienste mit 14% und im Bereich Computergrafik mit 11% verzeichnet. Die Anwendungsdomäne des maschinellen Lernens belegte mit 8% den dritten Platz, neben den Bereichen eingebettete Systeme, Simulation, Modellierung und Bioinformatik. MB beruhen auf Trainings- oder Testdatentransformationen, die zu einer gleichbleibenden oder erwarteten Veränderung der Vorhersageergebnisse führen. Nach [111] lassen sich die Datentransformationen in zwei verschiedene Ebenen unterteilen, die sich in ihrem Detailgrad unterscheiden. Die erste Ebene von Transformationen führt Änderungen mit einer erhöhten Granularität auf der Datenebene durch. Hierbei ändert die Transformation nicht eine einzelne Dateninstanz, sondern konzentriert sich auf den gesamten Datensatz. Dies ermöglicht grob granulare Änderungen, wie die Erweiterung des Datenbestands oder die Neuverteilung von Dateninstanzen. Im Gegensatz dazu führt die zweite Ebene Datentransformationen durch, indem sie Anpassungen an jeder einzelnen Dateninstanz vornimmt. Bei der fein granularen Datentransformation werden insbesondere Merkmale, Beschriftungen oder einzelne Bilderpixel verändert.

Murphy et al. [112] untersucht, welche Datentransformationen als MB eingesetzt werden können. Er identifizierte sechs Transformationen des Ausgangs Testfalls: addierend, multiplizierend, permutierend, invertierend, inklusiv und exklusiv. Addieren und Multiplizieren beschreiben die MB, bei der eine Änderung der Testausgangsdaten durch Addition oder Multiplikation mit einem konstanten Faktor für alle Daten im Datensatz keine Auswirkungen auf den

Testausgang haben sollte. Eine invertierende MB ist dann gegeben, wenn die Ausgabe vorhergesagt werden kann, indem das „Gegenteil“ der Eingabe verwendet wird. Die Begriffe „inklusiv“ und „exklusiv“ beschreiben die MB, dass das Hinzufügen und Entfernen von Daten zu einem Datensatz ein vorhersehbares Verhalten aufweisen.

Ding et al. [113] stellen in ihrem Artikel 11 MB vor, um biologische Zellbilder mit DL zu klassifizieren. Die Autoren propagieren die Verwendung von MRs zur Validierung des ML-Systems auf drei unterschiedlichen Ebenen. MB auf Systemebene werden gegen alternative Algorithmen des maschinellen Lernens getestet. Auf Datensatzebene basieren die MB auf Trainingsdaten- oder Testdatentransformationen, die sich nicht auf die Klassifizierungsgenauigkeit auswirken sollten. Die Entfernung einer Datenkategorie sollte nicht zu einer Veränderung der Klassifizierung der verbleibenden Kategorien führen. Die MB in der Datenelement-Ebene werden in Abhängigkeit von der Klassifizierungsgenauigkeit der reproduzierten Einzelbilder erstellt.

Xie et al. [114] schlagen vor, modellspezifische MB zu verwenden, um Implementierungen von überwachten Klassifikatoren zu testen. Die Studie umfasst fünf Arten von MB, die erwartete Änderungen an der Ausgabe (z.B. Änderungen in Klassen, Bezeichnungen, Attributen) auf der Grundlage spezifischer Änderungen an der Eingabe vorhersagen. MB, die aus Verhaltensweisen des Problems abgeleitet werden, können als notwendige Eigenschaften des ML-Algorithmus betrachtet werden. Die Untersuchung zeigt, dass nicht alle abgeleiteten MB notwendig und spezifisch für das gewählte ML-Modell sind. Die Notwendigkeit von MB wurde ebenfalls von Al-Azani [115] untersucht. Die in dieser Veröffentlichung untersuchten und miteinander verglichenen ML-Algorithmen sind: NB, k-NN und ihr jeweiliger Ensemble-Klassifikator[4]. Hierbei haben die Autoren festgestellt, dass eine MB, die für NB und k-NN notwendig ist, nicht unbedingt für ihren kombinierten Ensemble-Klassifikator erforderlich ist.

Im Gegensatz zu den oben genannten Verfahren konzentriert sich Xie et al. [116] auf einen metamorphen Testansatz für die Validierung von unüberwachten Lernprozessen. METTLE ist eine Abkürzung für „METamorphic

[4] Die Ensemble-Methode ist eine Technik des maschinellen Lernens, die mehrere Basismodelle kombiniert, um ein einziges Vorhersagemodell zu erstellen, das den einzelnen Basismodellen möglichst überlegen ist.

Testing approach to assessing and validating unsupervised machine LEarning systems“.METTLE formuliert 11 generische MB, die Eigenschaften abdecken, die von den Nutzern erwartet werden und die unüberwachte Clusteralgorithmen besitzen. Durch diese MBen werden die Dimensionen der Cluster beeinflusst, Datenpunkte an Clustergrenzen hinzugefügt oder die Dichte von einem oder mehreren Clustern verändert.

4.3.2 Anwendungsbereiche und Metamorphe Beziehungen im maschinellen Lernen und anderen Domänen

In einer Reihe relevanter Studien haben Zhou et al. [117–119] die Methodik des metamorphen Testens angewendet, um Diskrepanzen in Online-Suchdiensten zu identifizieren. In ihrer Untersuchung [117] realisierten die Forscher eine empirische Analyse der Suchmaschinen Google, Bing, dem chinesischen Bing sowie Baidu. Weiterhin führten Zhou et al. in den Publikationen [118] und [119] das Konzept der „generalisierten metamorphen Beziehung“ ein. Diese abstrahierte Form der metamorphen Beziehung dient als Grundlage für die Entwicklung von MB und erweitert somit das Spektrum der Anwendbarkeit des metamorphen Testens. Aufbauend auf dem eingeführten Konzept formulierten Segura et al. [120] sechs abstrakte Beziehungen in Datenbanken, die sie als Metamorphic Relation Output Patterns[5] (MROPs) bezeichnen. Jedes MROP beschreibt eine abstrakte Ausgaberelation, die in Web-APIs zu finden ist, unabhängig von deren Anwendungsbereich (siehe Abbildung 4.6).

Die in Kapitel Unterabschnitt 9.1.1 vorgestellten Muster Metamorpher Cluster Beziehungen (MMCB)s unterscheiden sich in wesentlichen Punkten von den erwähnten wissenschaftlichen Publikationen von Zhou und Segura et al. [117, 120]. Anstelle der Fokussierung auf die Identifizierung von Inkonsistenzen in Web-APIs richtet sich das Interesse dieser Dissertation auf die Aufdeckung von Fehlern in SSIB, im Hinblick auf die Analyse des Benutzerverhaltens im Kontext der Automobilindustrie.

Diese thematische Neuausrichtung erfordert eine Anpassung sowohl in Bezug auf die Art des Testinputs als auch der Testoutputs. Im Unterschied zu den Untersuchungen von Zhou und Segura et al. [117, 120], die sich vornehmlich

[5] zu Deutsch: Metamorphe Relationsausgabemuster

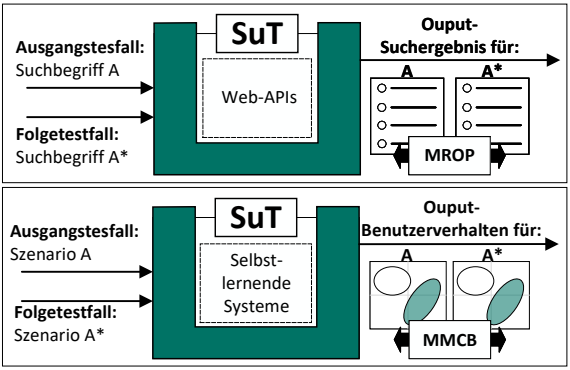

Abbildung 4.6: Differenzierung zwischen MROPs nach Segura et al. [120] und den in dieser Dissertation spezifizierten MMCBs (siehe Unterabschnitt 9.1.1)

auf die Analyse von Suchergebnissen als Antwort auf definierte Suchbegriffe konzentrieren, fokussiert die vorliegende Dissertation die Exploration von Szenarien, welche das adaptierte Benutzerverhalten infolge der Interaktion mit selbstlernenden Systemen abbilden. In diesem Kontext werden Cluster von Benutzeraktionen betrachtet, um die Muster der Interaktion und ihre Entwicklung über die Zeit hinweg zu analysieren.

Die Testsoftware mit dem Namen DeepTest [121] betrachtet KNNs für die Bilderkennung. Die Forscher haben drei verschiedenartige Modelle tiefer KNNs für das automatisierte Fahren untersucht. Der Schwerpunkt liegt dabei auf der Untersuchung von Grenzfällen, die ein falsches Fahrverhalten aufweisen, das zu potenziell tödlichen Kollisionen führen kann. Um effektive Daten für die Testdurchführung für automatisierte Fahrsysteme zu erzeugen, verwendet DeepTest [121] einen suche-basierenden Greedy-Algorithmus, der neun verschiedene realistische Bildtransformationen durchgeführt: Änderung der Helligkeit, Änderung des Kontrasts, Verschiebung des Bildbereichs, Skalierung des Bildbereichs, horizontale Scherung des Bildbereichs, Drehung des Bildbereichs, Hinzufügen von Unschärfe, Hinzufügen von Nebel- und Regeneffekte. Zur Überprüfung der Korrektheit der Ergebnisse wurden MB, die auf den genannten Bildtransformationen basieren, als Testorakel verwendet. Mithilfe dieser MB können realistische synthetische Bilder auf der Grundlage von Ausgangsbildern erzeugt werden, die reale Phänomene simulieren.

Sun et al. [122] entwickelten ein Framework, TransRepair, welches für die vollautomatische Überprüfung und Reparatur der Konsistenz von maschinellen Übersetzungssystemen verwendet werden kann. TransRepair kombiniert Mutation von Wortpaaren mit MT zur Aufdeckung von Inkonsistenzfehlern, ohne dabei auf menschliche Testorakel zurückgreifen zu müssen. Ein Beispiel für einen Inkonsistenzfehler zeigt die Übersetzung mithilfe von Google Translate[6]. Der Satz: „Men do good in computer science“ wird als „Männer sind gut in der Informatik“ übersetzt. Anschließend wird der Eingangssatz verändert zu „Women do good in computer science“ Der metamorphe Test besteht nun darin, die beiden Übersetzungen der genannten Beispielsätze miteinander zu vergleichen. Entgegen der Erwartung wird der zweite Satz, mit Frauen im Subjekt, als „Frauen tun Gutes in der Informatik“ und nicht als „Frauen sind gut in der Informatik“ übersetzt, was einen Fehler in der Konsistenz darstellt.

In der Forschung zu metamorphen Testverfahren liegt der Fokus überwiegend auf der Identifikation von „effektiven“ MB, während die Generierung von „effektiven“ Testfällen bislang eine untergeordnete Rolle spielte. Eine beträchtliche Anzahl dieser Studien greift auf die Methode der zufälligen Erstellung von Quelltestfällen zurück [123, 124].

Myers [125] kritisiert diese Herangehensweise als die am wenigsten effiziente Methode. Er argumentiert, dass eine Menge zufällig ausgewählter Testfälle nicht das Potenzial hat, optimal oder nahezu optimal zu sein, insbesondere im Hinblick auf die Wahrscheinlichkeit, eine maximale Anzahl an Fehlern zu identifizieren. Angesichts dieser Kritik wurden Versuche unternommen, durch den Einsatz fortschrittlicherer Techniken wie dem Adaptives Random Testing (ART) [123, 126] eine Verbesserung zu erzielen. ART ist eine erweiterte Form des zufälligen Testens, die darauf abzielt, die Effizienz und Effektivität bei der Fehlerentdeckung in Softwaretests zu verbessern. Im Gegensatz zum traditionellen zufälligen Testen, bei dem Testfälle rein zufällig aus dem gesamten Eingabebereich ausgewählt werden, nutzt ART heuristische Methoden, um die Auswahl der Testfälle zu steuern. Diese Heuristiken basieren auf der Annahme, dass Fehler in der Software in Clustern auftreten. Die bisherigen Forschungsarbeiten, die solche Ansätze verfolgen, befinden sich allerdings noch in einem frühen Entwicklungsstadium und beruhen weiterhin auf der Generierung von zufälligen Testfällen.

[6] Die beiden Übersetzungen wurden am 11.03.2024 erstellt.

4.4 Diskussion des Standes der Technik im Vergleich zur aktuellen Forschung

Die Validierung selbstlernender Funktionen, die individuelle Benutzeraktionen einbeziehen, erweist sich bei der Entwicklung abgeleiteter Testorakel als herausfordernd. Unter Einbeziehung der in Abschnitt 1.2 dargelegten initialen Anforderungen sowie die in Abschnitt 3.4 eingeführten Bewertungskriterien erfolgt eine Bewertung der verschiedenen Methoden (siehe Abbildung 4.7-Abbildung 4.9). Diese Evaluation zielt darauf ab, die Anwendbarkeit der Methoden für die Validierung selbstlernender Systeme, die auf Benutzeraktionen basieren, zu bestimmen.

4.4.1 Bewertung abgeleiteter Testorakel in Form von Kreuzreferenzierungen

Differenzielles Testen

Differenzielles Testen (siehe Unterabschnitt 4.1.1) verwendet als ATO eine alternative Implementierung des SuT. Diese Implementierungen werden von verschiedenen Entwicklern oder in unterschiedlichen Programmiersprachen erstellt. Durch die parallele Ausführung mehrerer Versionen und den Vergleich ihrer Ergebnisse wird die Zuverlässigkeit erhöht, was die Wahrscheinlichkeit eines Systemfehlers aufgrund von Softwarefehlern verringert.

Die parallele Entwicklung und Wartung mehrerer Softwareversionen führt jedoch zu erhöhtem Aufwand, da der Ressourcenbedarf und der zeitliche Einsatz im Vergleich zur Bearbeitung einer einzigen Version ansteigen. Eine weitere Limitation des differenziellen Testens ist die eingeschränkte Fehlerabdeckung, insbesondere bei Spezifikationsfehlern, da die Verwendung identischer Spezifikationen durch alle Entwicklungsteams eine gemeinsame Anfälligkeit für Fehler aufweist. Ein Nachteil des differenziellen Testens sind falsch positive Ergebnisse. Diese treten auf, wenn verschiedene Softwareversionen unterschiedliche Ergebnisse liefern, die jedoch nicht auf Programmierfehler zurückzuführen sind. Stattdessen resultieren sie aus unterschiedlichen, aber korrekten Interpretationen derselben Spezifikationen. Solche Diskrepanzen können Mehraufwand verursachen, da sie eine Fehlersuche auslösen, obwohl kein tatsächlicher Fehler vorliegt. Differenzielles Testen wird unter anderem ver-

wendet, um DL-Bibliotheken zu validieren oder um selbstlernende Systeme zu testen. Ein unterschiedliches Verhalten der n Implementierungen detektiert einen Fehler in mindestens einer der implementierten Version. Diese Methode ist für SSIB allerdings nicht vorteilhaft, da die Individualität des Nutzerverhaltens dazu führt, dass ein abweichendes Verhalten in der Funktion nicht als Fehler interpretiert werden soll, sondern vielmehr eine erwünschte Funktionseigenschaft darstellt.

Regressionstests

Regressionstests (siehe Unterabschnitt 4.1.2) stellen einen Schritt im Softwareentwicklungsprozess dar, mit dem Ziel zu überprüfen, dass Modifikationen oder Updates der Software nicht zu unbeabsichtigten Nebeneffekten in den bereits vorhandenen Funktionen führen. Die ursprüngliche Version der Software fungiert hierbei als ATO, gegen die neue Versionen getestet werden, um festzustellen, ob und welche Verhaltensänderung durch die vorgenommenen Softwareänderungen entstehen. Für SSIB kann ein Update als eine Veränderung des Benutzerverhaltens angesehen werden. Ein Vergleich im Sinne des Regressionstests ist damit obsolet, da die Softwareversion mit dem neuen Benutzerverhalten das korrekte Verhalten darstellt und die Referenzversion ein veraltetes Verhalten abbildet.

Kreuzreferenzierungen in Form von differenziellen Tests und Regressionstests dienen dazu, Beziehungen und Abhängigkeiten zwischen verschiedenen Teilen eines Systems oder Implementierungen zu identifizieren. Sie ermöglichen es, die Auswirkungen von Änderungen im Programmcode zu verstehen. Allerdings kann sich das dynamische Verhalten und die personalisierten Funktionen, die sich aus Benutzeraktionen ergeben, ändern. Diese Variabilität erfordert flexible und adaptive Testansätze, die sich an neue Verhaltensweisen und Benutzerpräferenzen anpassen können. Diese Anpassungsfähigkeit geht über die starren Rahmenbedingungen traditioneller Kreuzreferenzierungen hinaus.

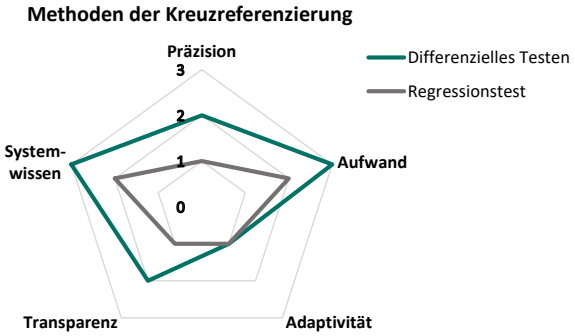

Abbildung 4.7: Darstellung und Vergleich von differenziellem Testen und Regressionstests im Kontext von Kreuzreferenzierungen

4.4.2 Bewertung abgeleiteter Testorakel in Form von Spezifikationsmining

Statische Analyse

Die statische Analyse (siehe Unterabschnitt 4.2.1) im Spezifikationsmining untersucht den Programmcode ohne Ausführung, um Invariante und Strukturen zu identifizieren, die für die automatische Erzeugung von Spezifikationen für Softwarekomponenten genutzt werden können. Ein Vorteil dieser Methode ist die Möglichkeit zur frühzeitigen Fehlererkennung, da sie es erlaubt, Fehler im Programmcode bereits in den Anfangsphasen der Entwicklung zu identifizieren, bevor der Programmcode tatsächlich ausgeführt wird.Der Hauptnachteil der statischen Analyse beim Testen einer SSIB liegt in ihrer begrenzten Fähigkeit, die dynamischen und datengetriebenen Aspekte von ML-Modellen und -Algorithmen zu erfassen. Dieser Nachteil hat die folgenden Auswirkungen:

- Limitationen bei der Beurteilung der Korrektheit der durch das ML-Modell generierten Testausgaben, da die Qualität neben der Wahl des Modells von den verwendeten Trainingsdaten abhängt.
- ML-Modelle, die aus Nutzeraktionen lernen und sich im Laufe der Zeit anpassen, entziehen sich der Fähigkeit statischer Analysen, ihr Verhalten bei zukünftigen, unbekannten Eingaben vorherzusagen.

- Die dynamische Lern- und Anpassungsfähigkeit von ML-Modellen an neue Nutzeraktionen kann von statischen Analysemethoden, die Ausführungskontexte oder -daten nicht einbeziehen, nicht bewertet werden.

Dynamische Analyse

Die dynamische Analyse (siehe Unterabschnitt 4.2.2) unterscheidet sich von der statischen Analyse durch ihre Fähigkeit, Laufzeitinformationen zu berücksichtigen. Dies ermöglicht es ihr, das dynamische Verhalten eines Systems zu beobachten und zu analysieren. In diesem Kontext haben sich ML-basierte Methoden als wertvoll erwiesen. Durch die Analyse von Testdaten können diese Verfahren eine alternative Version des SuT erlernen, was in der aktuellen Forschung zunehmend genutzt wird.Diese Herangehensweise bietet den Vorteil, dass Muster und Verhaltensweisen aus den Daten gelernt werden, wodurch tiefe Einblicke in die Funktionsweise des SuT gewonnen werden können. Diese Funktionsweise kann als Testorakel dienen.

Allerdings bringt diese Technik spezifische Herausforderungen mit sich. Ein zentraler Nachteil ist die Datenabhängigkeit: die Varianz der verwendeten Testdaten sind entscheidend für den Erfolg. Unzureichende oder verzerrte Datensätze können zu einem unvollständigen oder fehlerhaften Verständnis des SuT führen. Zudem stellen ML-Modelle, insbesondere tiefe neuronale Netze, „Black Boxes“ dar, deren Entscheidungsfindungsprozesse nicht betrachtet und somit interpretiert werden kann. Diese mangelnde Interpretierbarkeit erschwert nicht nur die Identifizierung von Fehlern, sondern die Begründung von Entscheidungen, die das ML-Modell trifft. Im Kontext der Validierung selbstlernender Systeme, die auf individuellen Benutzeraktionen basieren, stellt die Gewinnung repräsentativer Datensätze, die diese Aktionen adäquat widerspiegeln, eine Herausforderung dar. Die inhärente Dynamik individueller Benutzeraktionen verlangt nach fortgeschrittenen Methoden der Datenerfassung und -analyse, um ein zuverlässiges Testorakel für SSIB zu können. Ferner bedeutet die Anwendung der dynamischen Analyse, dass die aus dem Lernprozess resultierenden Spezifikationen spezifisch für das Verhalten einzelner Benutzer sind und bei Änderungen im Nutzerverhalten entsprechend aktualisiert werden müssen, um ihre Gültigkeit zu bewahren.

Spezifikationsmining erweist sich als effektive Methode zur Generierung von Spezifikationen bei regelbasierten Algorithmen. Allerdings stoßen traditio-

nelle statische Analyseverfahren bei selbstlernenden Systemen an ihre Grenzen, hauptsächlich aufgrund der datengetriebenen Natur dieser Systeme (siehe Abbildung 4.8). Dynamische Analysemethoden bieten sich als Lösung an, um selbstlernende Systeme effektiv zu testen, da sie auf der Untersuchung des tatsächlichen Verhaltens beruhen. Die Anwendung solcher Methoden auf Funktionen, die auf individuellen Benutzeraktionen basieren, ist jedoch nicht geeignet. Die Notwendigkeit, Spezifikationen individuell für jeden Benutzer zu extrahieren und diese bei jeglicher Verhaltensänderung zu aktualisieren, stellt einen unverhältnismäßig hohen Aufwand dar.

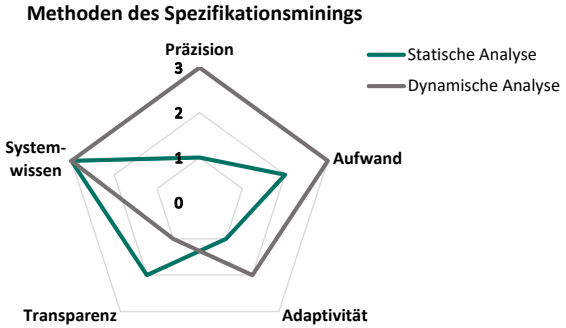

Abbildung 4.8: Darstellung und Vergleich von statischer und dynamischer Analysemethoden im Spezifikationsmining

4.4.3 Bewertung des Metamorphen Testens

Aufgrund seiner Datenorientierung zeigt sich metamorphes Testen wirkungsvoll bei der Bewertung von Systemreaktionen auf variierende Eingabedaten. Durch die Überprüfung der Beständigkeit von Verhaltensmustern oder Eigenschaften unterstützt es eine fundierte Analyse der Systemantworten auf diverse Datensätze. Für SSIB, die sich kontinuierlich weiterentwickeln und auf neue Benutzeraktionen einstellen, bietet MT essenzielle Vorteile. Es ermöglicht die fortlaufende Leistungsüberprüfung durch die Erstellung angepasster Folgetestfälle, die auf aktuellen Änderungen beruhen. Diese Methode adressiert effektiv die Limitationen konventioneller Testverfahren, die nicht mit der fortschreitenden Entwicklung des Systems mithalten können, insbesondere wenn Benutzeraktionen im Fokus stehen. Durch die Definition metamorpher Beziehun-

gen erleichtert das MT zudem die Entdeckung unerwarteter Verhaltensweisen oder Anomalien in der Systemreaktion auf individuelle Benutzeraktionen. Die Wirksamkeit des Metamorphen Testens ist eng mit der Qualität und Repräsentativität der Eingabedaten verknüpft; mangelhafte oder verzerrte Datensätze können zu verfälschten Testergebnissen führen.

Die Entwicklung relevanter metamorpher Beziehungen, die die Beziehungen zwischen Eingaben und Ausgaben treffend darstellen, bleibt für Systeme, die für individuelle Benutzeraktionen lernen, eine Herausforderung. Während ein Großteil der Forschung auf das Identifizieren von MB fokussiert ist, erhält die Generierung geeigneter Ausgangstestfälle vergleichsweise wenig Aufmerksamkeit [108]. Diese Vernachlässigung birgt das Risiko, dass nicht alle spezifischen oder einzigartigen Benutzeraktionen sowie die daraus resultierenden Lernprozesse vollständig erfasst werden. MT stellt eine effiziente Methode dar, um Systeme mit dynamischen und personalisierten Eigenschaften zu evaluieren. Die methodische Erzeugung qualitativ hochwertiger synthetischer Testdaten sowie die Identifikation adäquater metamorpher Beziehungen sind hierbei Voraussetzungen, deren Erfüllung eine Herausforderung darstellt. Daher wird MT unter den betrachteten Alternativen als die geeignetste Testmethode betrachtet, wodurch die Anforderung AM1 erfüllt wird.

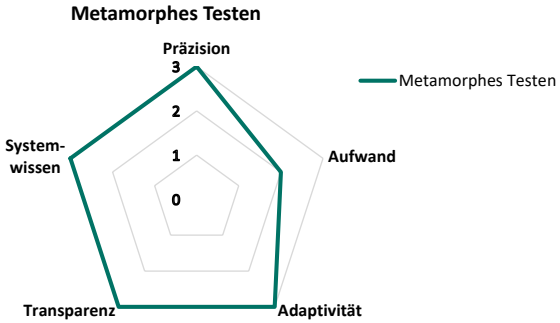

Abbildung 4.9: Netzdiagramm zur Bewertung des Metamorphen Testens nach den Kriterien Präzision, Aufwand, Adaptivität, Transparenz und Systemwissen.

5 Konzept zur Validierung selbstlernender Systeme

Die wissenschaftliche Fragestellung dieser Dissertation ist die systematische Erforschung von Validierungsmethoden für selbstlernende Systeme, mit einem Fokus auf individuell variierende Benutzeraktionen (siehe Abschnitt 1.3). Aus der Analyse in Kapitel 3, in dem Testorakel und ihre Einsatzmöglichkeiten diskutiert wurden, sowie aus Kapitel 4, das sich mit verschiedenen Testmethoden auseinandersetzt, wurde die MT-Methodik für die adressierte Problemstellung identifiziert (siehe Unterabschnitt 4.4.3). Die Anwendung des MT auf SSIB bringt Vorteile mit sich, die insbesondere in folgenden Aspekten zum Ausdruck kommen:

- MT adressiert das Testorakelproblem, indem es auf MB zurückgreift. Diese MB definieren erwartete Verhaltensänderungen des Systems in Reaktion auf variierende Eingaben, wodurch eine direkte Vorhersage spezifischer Ergebnisse nicht notwendig ist.
- Die Flexibilität von MT, durch den Einsatz von MB, ermöglicht eine adaptive Reaktion auf die Unvorhersehbarkeit von Benutzeraktionen. Dieses Verfahren erlaubt eine Analyse der Anpassungsfähigkeit des Systems an variierende Benutzeraktionen.
- Durch die Integration neuer Konsistenzbedingungen in Form von MB kann die Testabdeckung substanziell erweitert und eine Validierung der Systemfunktionalität sichergestellt werden.

Die bestehende Forschung hat sich bislang nicht mit der Anwendung von MT auf selbstlernende Systeme, die auf individuellen Benutzeraktionen basieren, auseinandergesetzt (siehe Unterabschnitt 4.3.1, Unterabschnitt 4.3.2). Insbesondere fehlt es an Erkenntnissen darüber, wie MT eingesetzt werden kann, um die Herausforderungen durch die Unvorhersehbarkeit von Benutzeraktionen zu

adressieren. Vor diesem Hintergrund erforscht die vorliegende Dissertation die Verwendung von Szenarien, um die identifizierte Forschungslücke von MT für selbstlernende Systeme, die auf individuellen Benutzeraktionen basieren, zu adressieren. Die folgenden Abschnitte diskutieren die Rolle von Szenarien im Detail: Die Effizienz des MT ist mit den eingesetzten MB sowie der Qualität der Ausgangstestfälle verknüpft. Bisherige Forschungsarbeiten konzentrierten sich primär auf die Rolle und Auswahl von MB hinsichtlich ihrer Kapazität zur Fehleraufdeckung, wie in Abschnitt 4.3 erörtert. Die Auswahl der Ausgangstestfälle ist von Bedeutung, insbesondere für die Validierung von Systemen, die darauf ausgelegt sind, realistisches Benutzerverhalten zu erlernen. Der aktuelle Stand der Technik tendiert dazu, Ausgangstestfälle zufällig zu generieren (siehe Kapitel 4).

Segura et al. [127] stellten fest, dass 57% der in Bezug auf MT untersuchten Publikationen auf zufällig erzeugte Ausgangstestfälle zurückgreifen, während 34% vorhandene Testfälle nutzen. Für das Testen eines SSIB ist das Erzeugen von zufälligem Verhalten (in Form von zufälligen Ausgangstestfällen) nicht zielführend, da dadurch kein menschenähnliches Verhalten überprüft wird, sondern rein zufällige Aktionen mit dem SuT.

Die Erzeugung zufälliger Testfälle, die nicht das Verhalten von Menschen nachahmen, begrenzt methodisch die Effektivität beim Testen selbstlernender Systeme, die auf Benutzeraktionen angewiesen sind. Solche Testfälle simulieren willkürliche Interaktionen mit dem SuT und reflektieren nicht die tatsächliche Nutzungsdynamik. Diese Dissertation zielt darauf ab, diese Forschungslücke durch die Integration eines szenariobasierten Ansatzes in das MT-Verfahren zu schließen.

Der gezielte Einsatz von Szenarien ermöglicht die Betrachtung realitätsnaher Ausgangstestfälle innerhalb des MT-Prozesses und verspricht somit eine präzisere Abdeckung des Testraums für SSIB, indem spezifische Benutzeraktionen und Verhaltensweisen in die Testkonzeption integriert werden.

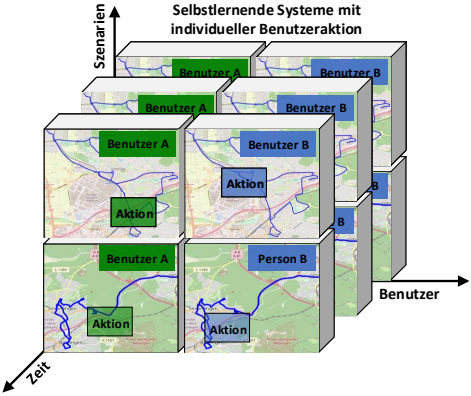

Abbildung 5.1: Testraum selbstlernender Systeme mit individuellen Benutzeraktionen. Jeder Würfel repräsentiert einen Ausgangstestfall, der durch ein bestimmtes Szenario definiert ist.

5.1 Testraum für selbstlernende Systeme mit individuellen Benutzeraktionen

Der Testraum (siehe Abbildung 5.1) bezeichnet den Gesamtraum aller möglichen Zustände und Interaktionen, die für die Validierung eines selbstlernenden Systems relevant sind. Dabei umfasst er alle Kombinationen von Szenarien und Benutzeraktionen, die das System im Laufe seiner Anwendung durchlaufen kann. Die verwendung eines formalisierten Testraums (siehe Abbildung 6.1) ermöglicht eine systematische und umfassende Prüfung des Verhaltens des Systems.

Im Fall eines SSIB umfasst der Testraum drei Dimensionen:

- Die erste Dimension erfasst das Spektrum unterschiedlicher Verhaltensweisen, die von Benutzern des SuT gezeigt werden.
- Die zweite Dimension repräsentiert diese Verhaltensweisen in verschiedenen Szenarien.

- Die dritte Dimension erweitert den Testraum, indem sie die zeitliche Entwicklung der Funktionen des selbstlernenden Systems berücksichtigt. Das Verhalten desselben Benutzers kann sich im identischen Szenario zu verschiedenen Zeitpunkten ändern, was die Festlegung eines korrekten Verhaltens durch ein Testorakel erschwert.

Die szenariobasierte Methode ermöglicht nicht nur eine Beschreibung und Abdeckung des Testraums für selbstlernende Systeme, sondern trägt dazu bei, die Erstellung von MB zu vereinfachen und zu formalisieren. Die in Abschnitt 4.3 diskutierten Methoden basieren auf generischen MB, die primär Datentransformation abdecken und universell über verschiedene SuT hinweg einsetzbar sind. Diese Ansätze sind jedoch für die spezifischen Anforderungen von SSIB, welche das Verhalten von Benutzern erlernen sollen, nicht ausreichend präzise. Insbesondere fehlt es an der notwendigen Spezifität, um das individuell abgestufte Benutzerverhalten adäquat zu erfassen. Die in Abschnitt 4.3 vorgestellten Methoden greifen auf generisch konzipierte MB zurück, die sich auf Datentransformationen konzentrieren und generell auf alle Arten von SuT anwendbar sind.

Indessen zeigt sich für die Applikation von MT im Kontext von SSIB, welche spezifisch das Erfassen und Interpretieren von Benutzerverhaltensweisen als Zielsetzung verfolgen, dass die ausschließliche Fokussierung auf datenorientierte MB eine Limitation darstellt. Der integrative Einsatz von Szenarien erweist sich als förderlich für die Konzeption benutzerzentrierter MB, vornehmlich durch:

- **Relevanz der Benutzeraktion:** Szenarien bieten Beschreibungen spezifischer Nutzungskontexte eines Systems und spiegeln realitätsnahe Benutzeraktionen wider. Indem sie auf solch realistische Szenarien abzielen, ermöglichen sie die Identifizierung von MB, die für das tatsächliche Benutzerverhalten relevant sind – über theoretische Annahmen oder willkürliche Daten hinausgehend.
- **Diversität und Abdeckung:** Die Generierung von MB auf Basis von Szenarien fördert die Erfassung eines breiteren Spektrums an Benutzerverhalten. Dies führt zu einer umfangreicheren Testabdeckung, da Szenarien unterschiedliche Benutzer, Ausnahmesituationen und Randfälle berücksichtigen.

- **Präzision in der Testfalldefinition:** Szenariobasierte Testfallbeschreibung ermöglicht eine Definition von MB, die spezifische Zustände und Interaktionen des zu testenden Systems adressieren.

5.2 Validierung durch die Kombination von Szenarien und metamorphen Tests

Das in dieser Dissertation entwickelte und untersuchte Konzept, das die Kombination von Szenarien mit dem MT zur Validierung von selbstlernenden Systemen unter Berücksichtigung von Benutzeraktionen betrachtet, ist in Abbildung 5.2 visualisiert. Das Konzept wird aufbauend auf dem Systementwurf (siehe Abbildung 1.4) in die folgenden vier Bereiche aufgeteilt:

- **Szenariodefinition:** Das Konzept zur Kombination von Szenarien mit metamorphem Testen selbstlernender Funktionen mit Benutzeraktionen (siehe Abbildung 5.2) widmet sich im ersten Abschnitt der Identifikation relevanter Szenarien zur Überprüfung der Funktionalität der selbstlernenden Systeme. In diesem Zusammenhang erfolgt zunächst eine Anforderungsanalyse, mit dem Ziel, die für das zu untersuchende System relevanten Szenarien zu erkennen und sie in Form von funktionalen Szenarien (Unterunterabschnitt 2.3.1) zu definieren. Die methodische Strukturierung von funktionalen Szenarien gemäß den etablierten PEGASUS Richtlinien (siehe Unterabschnitt 2.3.2) ist primär auf die Präzisierung von Fahrfunktionen ausgerichtet und nicht auf Funktionen, die eine direkte Interaktion mit dem Nutzer einschließen. Angesichts dessen konzentriert sich dieser Bereich auf die Untersuchung der Forschungsfrage 1 „Testspezifikation“, welche die Entwicklung von wiederverwendbaren Testfällen für selbstlernende Funktionen mit eingebetteter Benutzeraktion thematisiert. Im anschließenden Schritt erfolgt die Ausgestaltung von Szenarien, entweder durch Expertenwissen oder unter Anwendung einer morphologischen Matrix. Das Resultat dieser Phase umfasst die Darstellung logischer Szenarien (siehe Unterunterabschnitt 2.3.1), die als Grundlage für die Generierung von Testfällen dienen.
- **Szenarioerzeugung:** Die im Vorfeld konzipierten logischen Szenarien werden im anschließenden Schritt definiert und in konkrete Szenarien (siehe Unterunterabschnitt 2.3.1) umgewandelt. Die Erstellung realitäts-

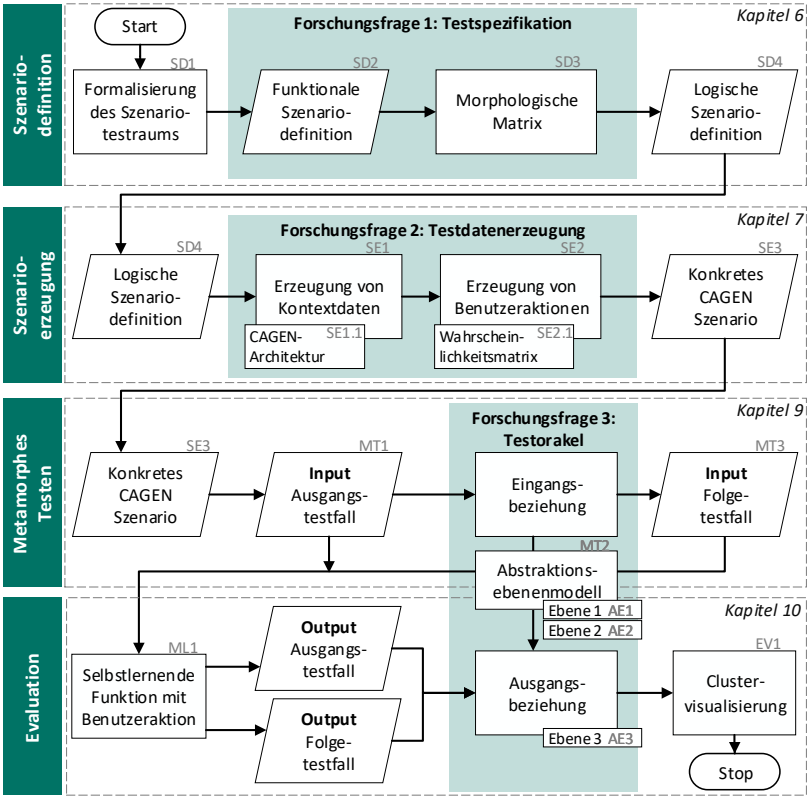

Abbildung 5.2: Darstellung des Konzepts zur Validierung von selbstlernenden Systemen mit Benutzeraktion durch die Kombination von Szenarien und metamorphen Testen. Die in Abschnitt 1.3 vorgestellten Forschungsfragen werden in den farblich hervorgehobenen Bereichen betrachtet.

naher Testdaten stellt dabei eine Herausforderung dar und bildet den Fokus der Forschungsfrage 2 „Testdatenerzeugung“. In diesem Kontext wurde das Werkzeug CAGEN entwickelt. Der Name CAGEN leitet sich aus dem englischen Akronym „Context and Action Generation“ ab. Die Software ermöglicht die Erzeugung von modularen Testdatensätzen durch ein Providerprinzip (siehe Abschnitt 7.1). Dabei werden Kontextdaten wie GPS-Koordinaten, Temperaturmessungen oder Zeitstempel, die für das Training und die Validierung von maschinellen Lernfunktionen eingesetzt werden können, erzeugt. Überdies befähigt CAGEN zur Erzeugung von Benutzeraktionen, die abhängig vom Kontext und abhängig von einem Benutzer auftreten können.

- **Metamorphes Testen:** In diesem Abschnitt erfolgt die Integration von Szenarien mit dem metamorphem Testen (siehe Abschnitt 4.3). Die im vorherigen Abschnitt „Szenariodefinition“ und „Szenarioerzeugung“ erzeugten konkreten Szenarien fungieren hierbei als Ausgangspunkt für die Durchführung zusätzlicher metamorpher Tests. Das MT stellt ein Verfahren dar, welches dazu dient, bestimmte erwünschte Eigenschaften des SuT zu überprüfen. Im Gegensatz zum szenariobasierten Testen liegt der Fokus hierbei nicht auf der Erfüllung oder Nichterfüllung einzelner Szenarien, sondern auf der Bewertung besonderer Eigenschaften des SuT. Die Durchführung von metamorphen Tests erfordert die Erzeugung von zwei sich unterscheidenden Testdatensätzen, die in einer Eingangsbeziehung zueinander stehen. Zur Bestimmung metamorpher Beziehungen wird das Abstraktionsebenenmodell (siehe Abbildung 9.1) eingeführt, das den Validierungsprozess für selbstlernende Systeme systematisch strukturiert. In Ebene 1 (siehe Unterabschnitt 9.1.1) werden typische Muster und Beziehungen identifiziert, die für die Validierung relevant sind. Ebene 2 (siehe Unterabschnitt 9.1.2) legt die Eigenschaften fest, die das SuT erfüllen muss, und definiert die spezifischen Aspekte, die durch das MT geprüft werden sollen. Für die Erzeugung des Input Folgetestfalls gemäß der definierten Eingangsbeziehung wird ebenso CAGEN eingesetzt[1].

[1] Der Einsatz von metamorphen Testmethoden erfordert die Möglichkeit zur synthetischen Datenerzeugung.

- **Evaluation**
  Die zuvor erzeugten Input-Datensätze werden zur Validierung eines SuT verwendet. Konkret werden drei Prototypen einer selbstlernenden Komfortfunktion betrachtet, die darauf abzielen, das individuelle Verhalten eines Fahrers zu erkennen und daraufhin eigenständige Handlungen auszuführen. Aufgrund der individuellen Natur der Benutzeraktionen jedes Fahrers erweisen sich herkömmliche Teststrategien wie das szenariobasierte Testen, das lediglich die Eingabe-Ausgabe-Beziehung überprüft, als unzureichend. Die daraus resultierende Herausforderung des Testorakelproblems (siehe Kapitel 1.2) wird durch die entsprechende Forschungsfrage 3 aufgegriffen und durch den Einsatz des MT gelöst. Die Ebene 3 des Abstraktionsebenenmodells konzentriert sich auf die Durchführung konkreter Eingangs- und Folgetestfälle, um die zuvor definierten Eigenschaften (Ebene 2) zu überprüfen und dabei die in Ebene 1 identifizierten Muster und Beziehungen zur Validierung des Systems zu nutzen. Durch die Analyse der multidimensionalen Datensätze erfolgt die Bewertung der MB mithilfe einer Clustervisualisierung (siehe Abbildung 10.1).

Das vorgestellte Konzept vereint die Vorteile des szenariobasierten und metamorphen Testens, um eine Validierung von SSIBs zu ermöglichen. Die entwickelten Methoden und Werkzeuge, wie CAGEN, ermöglichen die Generierung realistischer und vielfältiger Testdaten, die eine Validierung unter praxisnahen Bedingungen unterstützen. Durch die Verbindung von Szenarien und metamorphem Testen wird zudem das Testorakelproblem in selbstlernenden Systemen adressiert.

In den folgenden Kapiteln (siehe Kapitel 6 bis Kapitel 10) werden die Spezifikationen des vorgestellten Konzepts (siehe Abbildung 5.2) weiter behandelt. Zunächst wird die methodische Herangehensweise bei der Definition und Generierung von Szenarien beschrieben, gefolgt von einer detaillierten Betrachtung des metamorphen Testens im Kontext selbstlernender Systeme. Abschließend werden die Evaluationsergebnisse vorgestellt und die Auswirkungen auf die praktische Anwendung sowie die zukünftige Forschung diskutiert.

6 Szenariodefinition für selbstlernende Systeme mit Benutzeraktionen

Die Validierung von SSIB erfordert die Beschreibung von Testszenarien. Im ersten Abschnitt des Konzepts zur Validierung von selbstlernenden Systemen mit Benutzerinteraktionen (siehe Kapitel 5) wird der Fokus auf die Beschreibung des Testszenarioraums, durch Kontext (siehe Definition 6.2) und Benutzeraktionen (siehe Definition 6.4) gelegt.

> **Definition 6.2 Kontext (engl.: Context):** Im Bereich der SSIB bezeichnet Kontext alle Informationen die durch ein Szenario s beschrieben werden und beinhaltet den Benutzer u und die Zeit t. (nach [128])

In Abbildung 6.1 wird der Szenariotestraum einer selbstlernenden Komfortfunktion dargestellt (nach Abbildung 5.1). Innerhalb dieses Raumes sind Szenarien als s_y gekennzeichnet, die zu einem spezifischen Zeitpunkt t_z auftreten können. In dem dargestellten Modell repräsentiert der Szenarioraum eine Vielzahl von Kontexten, innerhalb derer Benutzer u_x eine spezifische Benutzeraktion B demonstriert. Aufbauend auf Definition 6.2 und [128] werden Benutzeraktionen wie folgt definiert:

> **Definition 6.4 Benutzeraktionen (engl.: Actions/Events):** Benutzeraktionen B sind kontextspezifische Aktivitäten (auch als Events bezeichnet), die ein Benutzer in Abhängigkeit vom (u_x, s_y, t_z) durchführt. Diese Aktionen sollen durch ein SSIB erlernt und automatisiert werden.

Das Verhalten B wird somit durch die Parameter u_x für den Benutzer, s_y für ein Szenario und t_z für einen Zeitpunkt bestimmt. Die Benutzeraktionen B

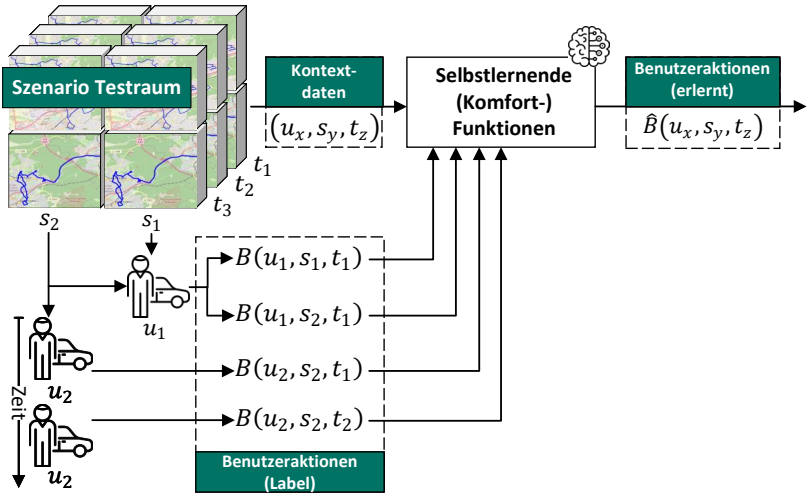

Abbildung 6.1: Szenariotestraum (nach Abbildung 5.1) und Formalisierung der Szenariodefinition. (siehe Abbildung 5.2, Bereich: SD1)

dienen als Label für eine selbstlernende personalisierte Funktion, die mithilfe der Kontextdaten, Muster aus den Benutzeraktionen extrahieren soll. Damit ergeben sich die folgenden Herausforderungen für die Validierung von SSIB bezogen auf den Szenariotestraum:

- Das erwartete Verhalten der Funktion ist von der Kombination der Parameter u_x, s_y und t_z abhängig. Die Anzahl an potenziellen Benutzer, Szenarien und Zeitpunkten geht gegen unendlich, sodass gilt:

$$(u_x, s_y, t_z) \rightarrow \infty \tag{6.1}$$

- Unterschiedliche Nutzer zeigen im identischen Szenario zu der identischen Zeit ein unterschiedliches Benutzerverhalten B.

$$B\,(u_x, s, t)\,! = B\,(u_{x+1}, s, t) \tag{6.2}$$

- Ein identischer Benutzer kann in verschiedenen Szenarien ein gleiches Benutzerverhalten B zeigen.

$$B\left(u, s_y, t\right) ! = B\left(u, s_{y+1}, t\right) \tag{6.3}$$

- Ein Benutzer kann sein Verhalten über die Zeit hinweg verändern, auch wenn die beeinflussenden Parameter (u_x, s_y) konstant bleiben.

$$B\left(u, s, t_z\right) ! = B\left(u, s, t_{z+1}\right) \tag{6.4}$$

6.1 Erweiterung der Szenariodefinition nach PEGASUS

Das Forschungsprojekt PEGASUS entwickelte eine Definition von Szenarien auf sechs Abstraktionsebenen. Hierbei lag das Hauptaugenmerk auf dem Bereich des hochautomatisierten Fahrens auf Autobahnen. Das Projekt entwickelte Definitionen und Beschreibungen der Szenarien, vorwiegend auf den technischen und kontextuellen Aspekten dieser spezifischen Fahrfunktionen. Zur Beschreibung von Szenarien für SSIB ist eine Erweiterung des PEGASUS-Modells um zusätzliche Ebenen erforderlich. Diese Erweiterung des bestehenden PEGASUS-Modells (siehe Unterabschnitt 2.3.2) erfüllt die Anforderung AS1. Vor diesem Hintergrund werden in dieser Dissertation zwei zusätzliche Ebenen vorgeschlagen und im Folgenden erläutert werden (siehe Abbildung 6.2).

- **Ebene 0 - Streckeninformationen** Auf dieser Abstraktionsebene erfolgt die Präzisierung der festzulegenden Ausgangs- und Endpunkte sowie der zu fahrenden Route. Diese Perspektive bietet eine makroskopische Sichtweise im Gegensatz zur bereits definierten Ebene 1, die sich auf die Straßenebene konzentriert und eine mikroskopische Sicht sowie eine Betrachtung der Fahrspuranordnung ermöglicht. Insbesondere für SSIB ist die Definition von Start- und Zielorten sowie der Fahrtroute relevant. Für SSIB ist es zudem von Bedeutung, den Kontext zu verstehen, in dem das Fahrzeug agiert. Start- und Zielorte geben Informationen über den geografischen Rahmen des Szenarios, während die festgelegte Route

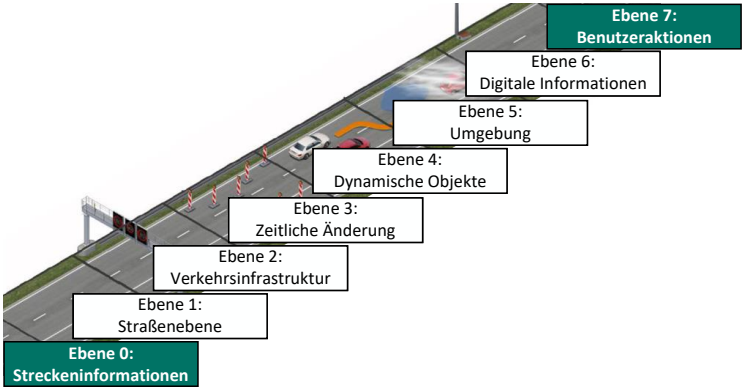

Abbildung 6.2: Visualisierung des Ebenenmodells des Forschungsprojektes PEGASUS [60] mit zwei zusätzlichen Ebenen für die Definition von Szenarien mit Benutzeraktionen. (siehe Abbildung 5.2, Bereich: Szenariodefinition, SD2.

den gewünschten Fahrweg definiert. Durch die Festlegung von Start- und Zielorten sowie der Fahrtroute auf dieser Ebene, kann eine grundlegende Struktur für das Szenario definiert werden, die als Ausgangspunkt für weitere Ebenen des Szenarienmodells dient.

- **Ebene 7 - Benutzeraktionen** Innerhalb des PEGASUS-Projekts lag der Schwerpunkt auf der Charakterisierung von hochautomatisierten Szenarien, in welchen die Wechselwirkung zwischen Fahrzeug und Nutzer nicht im Vordergrund stand. Das primäre Ziel bestand darin, Szenarien zu entwickeln, in denen das Fahrzeug weitgehend automatisiert agiert und hauptsächlich auf seine Umgebung reagiert, ohne unmittelbar mit dem Nutzer in Interaktion zu treten. In Anbetracht der thematischen Fokussierung wurde in der vorliegenden Dissertation eine Erweiterung des bestehenden Modells in Form einer Ebene zur Definition von Benutzeraktionen eingeführt, um potenzielle zukünftige Entwicklungen zu berücksichtigen, in denen solche Interaktionen von Bedeutung sein könnten. Die Einbeziehung der zusätzlichen Ebene zur Definition von Benutzeraktionen in die Szenariodefinition eröffnet erweiterte Gestaltungsmöglichkeiten.

Die Integration der Streckeninformations- sowie der Benutzeraktionsebene eröffnet neue Perspektiven für die Beschreibung von Szenarien. Im Bereich des automatisierten Fahrens ermöglicht die Definition von Szenarien, die die spezifischen Interaktionen zwischen Fahrzeug und Benutzer beinhalten. Beispiele hierfür sind das Anfordern einer Fahrt, das Eingeben eines Ziels oder ein Bewerten des Verhaltens eines automatisierten Fahrzeugs. Ferner können auch Szenarien für Komfortfunktionen durch die Hinzufügung dieser Ebenen beschrieben werden. Hierzu zählen unter anderem Funktionen zur individuellen Anpassung der Fahrzeugeinstellungen, zur Bereitstellung von Informationen oder zum Erkennen von Routinen des Fahrers. Die Beschreibung von Szenarien mit Streckeninformationen und Benutzeraktionen erweitert somit den Anwendungsbereich und die Genauigkeit der Szenariodefinition. Sie trägt dazu bei, eine detailliertere Abbildung der realen Fahrsituationen und Benutzeraktionen zu erreichen.

6.2 Merkmalsauswahl und Ausprägungsfindung durch morphologische Matrix

Im Kontext einer morphologischen Matrix bezeichnet eine Ausprägung eine konkrete Ausgestaltung oder Variante einer bestimmten Eigenschaft oder eines Parameters des Systems. Jede Eigenschaft, die in der Matrix dargestellt wird, kann mehrere Ausprägungen besitzen, die mögliche Zustände oder Werte dieser Eigenschaft repräsentieren. Durch die Kombination verschiedener Ausprägungen aller Eigenschaften lassen sich unterschiedliche Systemvarianten oder Lösungsansätze identifizieren und analysieren.

Die vorliegende Spezifikation teilt die Auswahl der Merkmale und die Bestimmung ihrer Ausprägungen zunächst in zwei Bereiche auf. Der erste Bereich betrifft Merkmale, die die Umgebung beschreiben und nicht von einem Benutzer abhängen. Diese Merkmale werden als Kontextparameter bezeichnet und sind der Streckentyp, die Jahreszeit, die Tageszeit, das Wetter und mögliche Störungen. Anhand dieser Auswahl werden die Testdaten durch Provider (siehe Abschnitt 8.2) erzeugt. Nach der Auswahl dieser Merkmale wird im nächsten Schritt für jedes Merkmal eine Auswahl an möglichen Ausprägungen definiert, um den Kontext der Testszenarien zu spezifizieren (siehe Tabelle 6.1).

Im zweiten Bereich der Merkmalsauswahl und Ausprägungsfindung wird auf den Fahrer und die Eigenschaften innerhalb des Fahrzeugs eingegangen. Bei der Identifizierung von relevanten Merkmalen wurden die Merkmale Geschlecht, Alter, Fahrerprofil, die Anzahl der Passagiere sowie ein Erschöpfungswert und ein Wert für die Empfindlichkeit gegenüber Temperatur identifiziert (siehe Tabelle 6.2). Die als benutzerspezifische Parameter bezeichneten Merkmale sind individuell und beschreiben die verschiedenen Eigenschaften des Fahrers. Die spezifischen Ausprägungen dieser Parameter simulieren die Wahrscheinlichkeit, mit der der Fahrer bestimmte Aktionen ausführt. Die Wahrscheinlichkeit für eine Fensteröffnung ist bei einem simulierten Benutzer mit dem Fahrerprofil „Allergiker“ geringer als bei dem Fahrerprofil „Raucher“ (siehe Abschnitt 8.3).

	Kontextparameter						
Streckentyp	Autobahn		Stadt		Überlandstraße		
Jahreszeit	Frühling		Herbst		Winter		
Tageszeit	Frühmorgen	Morgen	Vormittag	Mittag	Nachmittag	Abend	Nacht
Wetter	Sonne		Regen		Schnee		Wolken
Störungen	Tunnel		Stau		Ampeln		keine Störung

Tabelle 6.1: Morphologische Matrix zur Definition der Merkmale der Kontextparameter. (siehe Abbildung 5.2, Bereich: Szenariodefinition, SD3.2)

	Benutzerspezifische Parameter				
Geschlecht	Frau		Mann		
Alter	18-29		30-65		65+
Fahrerprofil	Standard	Raucher	Allergiker		Frischluft
Anzahl Passagiere	1	2	3	4	5
Erschöpfung	gering	mittel	hoch		
Temperaturempfindlichkeit	gering	mittel	hoch		

Tabelle 6.2: Morphologischen Matrix zur Definition benutzerspezifischer Parameter. (siehe Abbildung 5.2, Bereich: Szenariodefinition, SD3.2

Durch die Kombination aller identifizierten Ausprägungen der Merkmale der beiden Bereiche aus Tabelle 6.1 und Tabelle 6.2 sind 1.244.160 logische Szenarien verfügbar.

Für jede der logischen Szenariokombinationen wurden mithilfe des in Kapitel 7 beschriebenen Verfahrens zur Szenarioerzeugung zehn konkrete Szenarien

erstellt. Dieses Verfahren führte zur Erzeugung von insgesamt über 12,4 Millionen Testdatensätzen. Aufgrund der Anzahl an generierten Testdaten ist die individuelle Bearbeitung jedes einzelnen Szenarios jedoch nicht durchführbar. Unter der Annahme, dass die Bearbeitung[1] eines Szenarios 15 Sekunden in Anspruch nimmt, würde die Auswertung aller Szenarien eine Gesamtdauer von 2160 Tagen oder nahezu sechs Jahren beanspruchen. Die Anzahl der logischen Szenarien kann durch die Verwendung von Wegzwecken (siehe Unterabschnitt 6.2.1) und durch die Abhängigkeiten Unterabschnitt 6.2.2 zwischen den Ausprägungen der Merkmale reduziert werden.

6.2.1 Auswahl repräsentativer Szenarien durch Wegzwecke

Um der Herausforderung des Szenarioraums (siehe Abbildung 5.1) zu begegnen, wird die Identifikation repräsentativer Szenarien anhand von Wegzwecken vorgenommen. Dabei wurde der Ergebnisbericht der Mobilität in Deutschland [129] herangezogen, der eine Untersuchung des Bundesministeriums für Verkehr und digitale Infrastruktur aus dem Jahr 2017 darstellt und verschiedene Aspekte der MiD. Die Berücksichtigung der Ergebnisse des MiD-Berichts ermöglicht eine Auswahl von repräsentativen Szenarien, da diese Wegzwecke alltägliche Fahrten abdecken und somit relevante Fahrsituationen repräsentieren. Indem die Resultate des MiD-Berichts als Grundlage für die Auswahl der Wegzwecke herangezogen werden, lässt sich eine gezielte Testabdeckung realisieren, welche auf die Mobilitätsmuster und Anforderungen der Nutzer abgestimmt ist. Die Untersuchung identifizierte spezifische Wegzwecke und teilte sie in sechs Gruppen ein. Diese sechs Gruppen umfassen Fahrten in Bezug zu Arbeit, Dienstfahrten, Einkäufe, Erledigungen, Freizeitaktivitäten und Begleitfahrten. Die Verwendung der Wegzwecke erfüllt die Anforderung AS2.

Gemäß dem Ergebnisbericht MiD bezieht sich der Wegzweck **Arbeit** auf typische Pendlerfahrten, die täglich mit dem Fahrzeug vom Wohnort zur Arbeitsstätte und zurück stattfinden. Es handelt sich dabei um regelmäßige Fahrten, die von Berufspendlern durchgeführt werden. Es ist jedoch zu beachten, dass die Definition der „regelmäßigen beruflichen Wege“ von dieser Klassifizie-

[1] Die Bearbeitung umfasst sowohl das Training des maschinellen Lernmodells als auch die Evaluierung der Testszenarien, um festzustellen, ob sie erfüllt oder unerfüllt sind.

rung ausgeschlossen ist. Diese Kategorie umfasst bestimmte Berufsgruppen wie Lieferanten, Handwerker oder Postboten, die aufgrund ihrer beruflichen Tätigkeit mehrere Arbeitsorte anfahren müssen und somit von den üblichen Pendlerfahrten abweichen [129].

Die Kategorie **Dienstfahrt** bezieht sich auf eine vorübergehende Tätigkeit, die außerhalb der regulären Arbeitsstätte stattfindet, wobei die Mobilität jedoch nicht Bestandteil der Arbeitstätigkeit ist. Dienstfahrten sind ein Bestandteil von Berufen und können sowohl innerhalb als auch außerhalb des Landes stattfinden. Typische Anlässe für Dienstreisen sind Kundenbesuche, Tagungen, Messen und Marktsondierungen. Im Gegensatz zur Pendlerfahrt sind Dienstreisen in der Regel nicht täglich, sondern treten in unregelmäßigen Abständen auf. In der Mobilitätsforschung werden unter dem Wegzweck **Erledigung** verschiedene Fahrttypen zusammengefasst, die im Zusammenhang mit alltäglichen Erledigungen stehen. Dazu gehören unter anderem Arztbesuche, Fahrten zur Behörde, zur Bank, zur Post oder zu religiösen Einrichtungen [129]. Im Kontext der Verkehrsmobilität werden Fahrten, die dem Besuch von Freunden, Bekannten oder Verwandten sowie dem Besuch von Veranstaltungen oder Sportvereinen dienen, als **Freizeitfahrten** kategorisiert.

Begleitungen im Kontext von Autofahrten beziehen sich auf das primäre Ziel, Kinder, ältere oder hilfsbedürftige Menschen zu transportieren. Hierbei handelt es sich um eine unterstützende Mobilitätsform, bei der die Hauptaufgabe darin besteht, andere Personen zu befördern. Dies kann insbesondere der Transport der eigenen Kinder zur Schule oder zum Kindergarten sein, oder der Besuch von älteren Verwandten oder Freunden, die nicht mehr mobil sind und eine Begleitung benötigen. Dabei kann auch der Besuch von Ärzten oder Therapeuten als Begleitfahrt angesehen werden, wenn die zu begleitende Person selbst nicht mehr in der Lage ist, sich selbstständig fortzubewegen.

6.2.2 Abhängigkeiten der Ausprägungen der morphologischen Matrix

Bei der Auswahl der Ausprägungen existieren Kombinationen, die sich aufgrund logischer Unvereinbarkeiten gegenseitig ausschließen. Ebenso existieren Ausprägungen, die voneinander abhängig sind. Diese Abhängigkeiten wurden sowohl aus statistischen Datenquellen als auch aus persönlichen Erfahrungen und Annahmen abgeleitet. Im Folgenden werden einige dieser Abhängigkei-

ten beschrieben. Ein Beispiel für die Abhängigkeit zwischen Ausprägung und Wegzweck ist die Ausprägung *Tageszeit*. Der Wegzweck *Einkauf* schließt späte Uhrzeiten aus, da Einkäufe in der Regel nicht nachts getätigt werden. Ebenso verhält es sich mit dem Wegzweck *Erledigung*, der frühmorgens und nachts seltener vorkommt (siehe Abbildung 6.3). Diese Abhängigkeiten werden bei der Erzeugung der Szenarien durch die morphologische Matrix berücksichtigt.

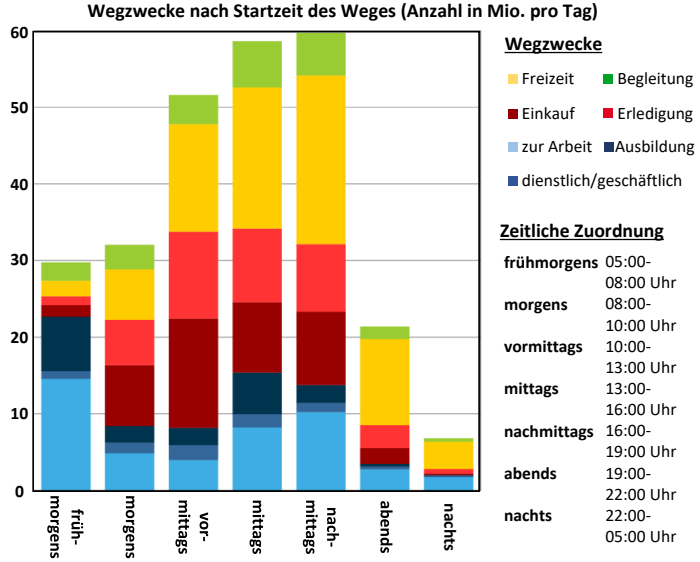

Abbildung 6.3: Wegzwecke nach Startzeit des Weges. (nach [129])

Die Anzahl der Passagiere im Fahrzeug variiert je nach Wegzweck. Es ist etwa unwahrscheinlich, dass fünf Personen im Fahrzeug unterwegs sind, wenn der Wegzweck *Pendeln* lautet. Der durchschnittliche Besetzungsgrad eines Fahrzeugs in Abhängigkeit vom Fahrtzweck wurde in einer Studie für das BMVI ([129]) ermittelt. Daraus ergibt sich, dass mit zunehmender Fahrtweite auch die Anzahl der Passagiere im Fahrzeug steigt. Das Passagierverhalten variiert ebenfalls je nach Wegzweck. Bei *Dienstfahrten* und *Einkäufen* ist das Verhalten eher statisch, da selten Personen ein- und aussteigen. Im Gegensatz dazu ist das Passagierverhalten bei *Freizeitfahrten* dynamischer, da Interaktionen mit Personen aus anderen Haushalten häufig zu Änderungen der Passagieranzahl

führen. Es besteht eine Abhängigkeit zwischen der Tageszeit und dem Wetter. Beispielsweise kann keine Kombination von Nacht und Sonne auftreten. Ebenso gibt es eine Abhängigkeit zwischen der Jahreszeit und dem Wetter. Im Sommer sind sonnige Tage häufiger, während im Winter Schnee wahrscheinlicher ist. Die Kombination von Schnee und Sommer wird ausgeschlossen. Der Streckentyp beeinflusst die Art der auftretenden Störungen. In städtischen Gebieten gibt es häufiger Ampeln, während auf Autobahnen Tunnel häufiger anzutreffen sind als in der Stadt. Unter Berücksichtigung dieser Abhängigkeiten werden die sechs verschiedenen Szenarien mithilfe der morphologischen Matrix erzeugt. Durch die Anwendung einer systematischen Szenariodefinition durch den Einsatz der morphologischen Matrix wird die Anforderung AS3 erfüllt. Die vollständige morphologische Matrix ist in Abschnitt 12.1 dargestellt. In Abbildung 6.4 sind drei Szenarien ausgewählt, die eine möglichst breite Abdeckung der morphologischen Matrix definieren. Die Wegzwecke „Begleitung“, „Erledigung“ und „Arbeit“ werden in Kapitel 10 weiter verwendet, um das SuT zu validieren.

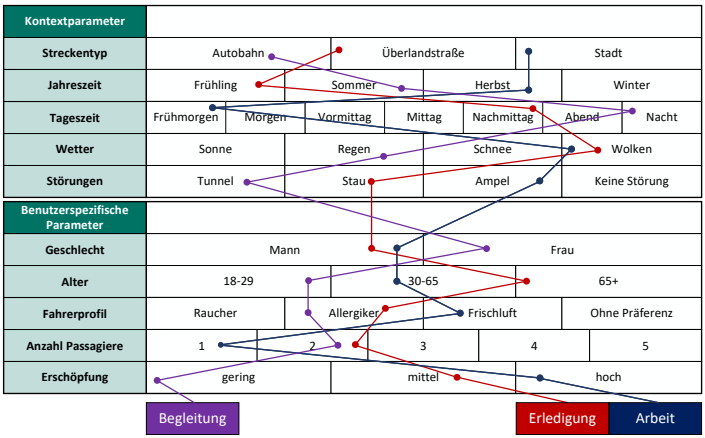

Abbildung 6.4: Kombination der Wegzwecke (Begleitung, Erledigung und Arbeit) gemäß MiD-Bericht [129] mit den Merkmalen der morphologischen Matrix zur Szenariodefinition (siehe Abbildung 5.2, Bereich: Szenariodefinition, SD3).

7 Szenarioerzeugung von Kontextdaten und Benutzeraktionen

Für SSIBs existieren zwei Methoden zur Erzeugung von Szenarien und der damit verbundenen Datengenerierung: die synthetische Erzeugung von Daten zur Sicherstellung einer ausreichenden Datenmenge sowie die Datenaugmentation zur Erhöhung der Datenvarianz.

Bei der Datenaugmentation wird die Variabilität der Daten durch Modifikation bestehender, realer Datensätze erweitert [130]. Insbesondere bei visuellen Datensätzen kommen Augmentationstechniken zum Einsatz, beispielsweise durch Rotationen, Anpassungen der Helligkeit (z.B. Über- oder Unterbelichtung) oder das Einbringen von simuliertem Sensorrauschen. Es ist zudem möglich, Sensordaten wie physikalische Messwerte in Zeitreihenformaten durch Augmentation zu modifizieren, um Variationen in den Daten zu erzeugen.

Im Gegensatz zur Augmentation werden zur Erhöhung der Datenmenge auch vollständig synthetische Daten erzeugt, die nicht auf existierenden realen Datensätzen basieren. Hierbei werden Softwaretools genutzt, die eine abstrahierte Darstellung der späteren realen Einsatzumgebung des Systems bieten [131]. Durch diese Softwaretools können relevante Kontextdaten (siehe Abbildung 8.9) generiert werden, die eine hohe Varianz aufweisen und auf die Anforderungen des SuT zugeschnitten sind, wie z.B. GPS-Rauschen oder zeitliche Variationen [111]. Im Vergleich zur Datenerfassung mit physischen Systemen bietet diese Methode der synthetischen Datenerzeugung höhere Effizienz und vermeidet gleichzeitig die Risiken und die möglichen Schäden, die bei realen Testfahrten auftreten können. Ein potenzieller Nachteil dieser Methode ist das Risiko der Einführung eines Bias, also einer systematischen Verzerrung oder Abweichung der simulierten oder erfassten Daten von den tatsächlichen Bedingungen, was zu einer ungenauen Darstellung oder Interpretation der realen Welt führen kann. [132]

In dieser Dissertation wird das Softwaretool CAGEN (siehe Kapitel 8) zur Erzeugung von synthetischen Testdatensätzen vorgestellt. CAGEN ermöglicht die Generierung von Kontextdaten aus modularen Datenquellen (siehe Abbildung 7.2) und integriert benutzerspezifische Interaktionen (siehe Abschnitt 7.4). Damit adressiert CAGEN die Forschungsfrage 2: „Wie lassen sich realitätsnahe Kontextdaten und benutzerspezifische Interaktionen synthetisch erzeugen?“ und erfüllt die Anforderung AD1.4. (siehe Abbildung 5.2).

7.1 Klassifikation und Spezifikation der Kontextkategorien

Um Szenarien entsprechend der Formalisierung (siehe Abbildung 6.1) realitätsnah zu simulieren und darauf basierend synthetische Testdaten zu generieren, wird der Kontext in drei Kategorien unterteilt. Diese Unterteilung ermöglicht es, die Einflüsse von Umweltbedingungen, Sensordaten und Benutzerinteraktionen separat zu simulieren und zu analysieren. Der Kontext wird daher in folgende Kategorien strukturiert: *Idealer-Kontext*, *Sensor-Kontext* und *Benutzer-Kontext* (siehe Abbildung 7.1).

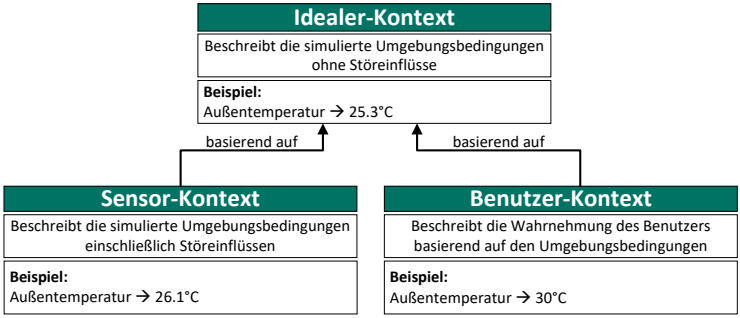

Abbildung 7.1: Aufteilung der simulierten Kontexte in drei unterschiedliche Bereiche. Der Sensor- und der Benutzer-Kontext basieren auf dem Idealwert des Idealen Kontexts und integrieren jeweils spezifische Einflüsse: Der Sensor-Kontext berücksichtigt externe Störungen, während der Benutzer-Kontext die Wahrnehmung des Benutzer widerspiegelt. (siehe Abbildung 5.2, Bereich: Szenarioerzeugung, SE1)

Idealer-Kontext: Dieser Kontext umfasst die simulierten Werte, die die Umweltbedingungen des Szenarios darstellen. Dazu gehören Werte wie GPS-Daten, Außentemperatur, Verkehrsdichte, Wetterbedingungen, Straßenverhältnisse und weitere für das SuT relevante Faktoren. Diese Parameter basieren auf den festgelegten Annahmen der definierten Szenarien durch Wegzwecke (siehe Abbildung 6.4) und werden von simulierten Datenquellen bereitgestellt (siehe Abbildung 7.3). Beispielsweise wird die simulierte Außentemperatur von 25,3°C im Idealen-Kontext durch den Temperatur-Provider (siehe Unterunterabschnitt 8.2) synthetisch generiert. Eine Einschränkung des Idealen-Kontexts ist jedoch, dass diese Werte keine Störeinflüsse wie Rauschen enthalten und somit die realen Bedingungen nicht vollständig widerspiegeln.

Sensor-Kontext: Der Sensor-Kontext adressiert diese Einschränkung, indem er die Werte des Idealen-Kontexts übernimmt und durch simuliertes Sensorrauschen oder zeitliche Variationen modifiziert. Der Begriff „Sensor-Kontext“ wird gewählt, da synthetische Daten auf Sensorebene simuliert werden und so die Erfassung durch verschiedene Sensorsysteme des Fahrzeugs abgebildet wird. Dazu gehören unter anderem Daten von Kameras, Radarsensoren, Lidarsensoren sowie weiteren Sensoren, die zur Detektion und Analyse der Umgebung genutzt werden. Die Einbeziehung des Sensor-Kontexts ermöglicht es, die Reaktionen des Fahrzeugs auf die simulierten Umgebungsbedingungen abzubilden. Durch das hinzugefügte Sensorrauschen kann ein Temperaturwert von 25,3°C im Idealen-Kontext im Sensor-Kontext beispielsweise als 26,1°C dargestellt werden.

Benutzer-Kontext: Analog zur Messung physikalischer Werte durch Sensoren bezieht sich der Benutzer-Kontext auf die Werte, die ein Benutzer interpretiert. Dieser Kontext entsteht auf Basis der Wahrnehmung des Idealen-Kontexts. Der Benutzer-Kontext wird durch benutzerspezifische Parameter der Szenariodefinition bestimmt, welche in der morphologischen Matrix festgelegt sind (siehe Tabelle 6.2). Ein Beispiel hierfür ist die individuelle Empfindlichkeit des Fahrers gegenüber Temperaturen, die sich auf die empfundenen Werte auswirkt. Im gegebenen Beispiel kann der Benutzer die durch den idealen Kontext simulierte 25,3°C Außentemperatur als 30,0°C wahrnehmen, aufgrund des individuellen Temperaturempfindens.

Die Unterteilung des Kontexts in die drei Kategorien – Idealer-Kontext, Sensor-Kontext und Benutzer-Kontext – ermöglicht die Simulation von Szenarien, die die relevanten Eigenschaften der Umgebung abbilden. Diese differenzierte Betrachtung ermöglicht eine Analyse und Bewertung der Leistung selbstlernender Systeme mit simulierten realitätsnahen Testdaten, wodurch die Anforderung AD1.4 erfüllt wird.

7.2 Mediator-Entwurfsmuster für anpassbare Testdatensätze

Nach der Beschreibung der Kontextaufteilung wird nun die Methodik zur Erstellung dieser Kontexte erläutert. Hierfür wurde das Softwaretool CAGEN entwickelt, das auf dem Mediator-Entwurfsmuster basiert.

> **Definition 7.2 Mediator-Entwurfsmuster:** Das Mediator-Entwurfsmuster ist ein Verhaltensmuster, das die Kommunikation zwischen Objekten über einen zentralen Mediator steuert. Dadurch werden direkte Abhängigkeiten reduziert und die Wartbarkeit des Systems erhöht. Jedes Objekt interagiert nur mit dem Mediator, der Informationen gezielt an die relevanten Objekte weiterleitet. So entsteht eine entkoppelte Struktur, in der Änderungen an einem Objekt keine Anpassungen bei anderen Objekten erfordern.

Im vorliegenden Mediator-Entwurfsmuster (siehe Abbildung 7.2) fungieren die zuvor genannten Objekte als Datenprovider, also Instanzen, die Daten über externe Schnittstellen bereitstellen. Um die verschiedenen Datenprovider zu kombinieren und einen flexiblen Testdatensatz zu generieren, wurde das Mediator-Entwurfsmuster modifiziert, um die Anforderungen AD2 zu erfüllen.

Datenprovider können entweder Daten über Dienste (APIs) generieren oder auf bereits vorhandene Datensätze zugreifen, die lokal oder extern in einem Cloudspeicher gespeichert sind. Jeder Datenprovider bietet eine unterschiedliche Anzahl von Eingangs- und Ausgangsdaten, wodurch mehrere Dienste gleichzeitig zur Datenerzeugung genutzt werden können. Die von den Diensten bereitgestellten Eingabedaten werden anschließend durch die interne Logik des Datenproviders transformiert. Die neu berechneten Daten bilden die Ausgabe

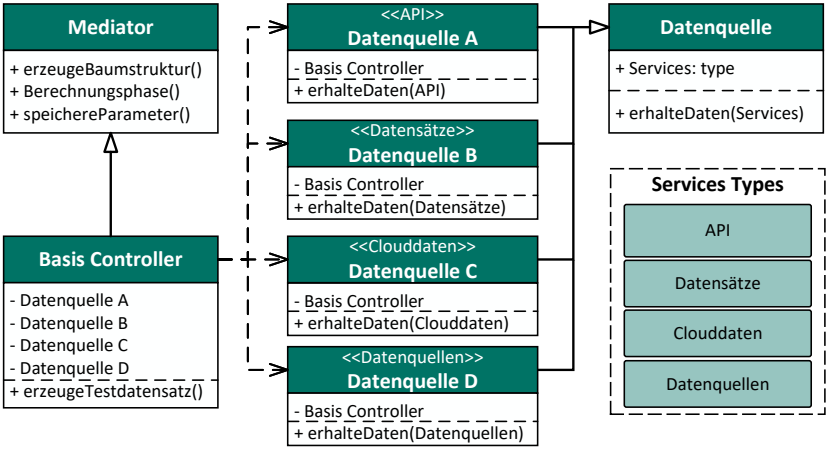

Abbildung 7.2: Angepasstes Mediator-Design-Pattern (siehe [SMS21]) für die Verwendung verschiedener Datenprovider und Erzeugung eines benutzerdefinierten Testdatensatzes (siehe Abbildung 5.2, Bereich: Szenarioerzeugung, SE1).

des Datenproviders und können von nachfolgenden Datenquellen weiterverwendet werden.

Der Basis-Controller übernimmt die Rolle des Mediators und legt die Reihenfolge fest, in der die einzelnen Datenprovider berechnet werden. Er steuert die Datenerzeugung und stellt sicher, dass keine unautorisierten Zugriffe oder Änderungen durch nicht autorisierte Datenprovider erfolgen. Der Betrieb des Basis-Controllers ist in zwei Phasen unterteilt: die **Vorbereitungsphase** und die **Evaluierungsphase**.

In der Vorbereitungsphase werden die notwendigen Datentabellen (siehe Tabelle 8.1) initialisiert. Zu diesem Zeitpunkt ist der Tabellenkörper nicht befüllt, da noch keine synthetischen Daten erzeugt wurden. Die Tabellenstruktur, also der Speicherbereich für die Daten, ist jedoch bereits festgelegt, einschließlich Parametern wie Abtastrate und Spaltenbeschriftungen.

Anschließend werden die Datenprovider basierend auf ihren Abhängigkeiten in eine logische Berechnungsreihenfolge gebracht. Dabei wird sichergestellt, dass die Berechnungen eines vorhergehenden Datenproviders abgeschlossen sind und die Ergebnisse für nachfolgende Berechnungen bereitstehen, bevor die

nächste Berechnung beginnt. Diese Abhängigkeiten bilden eine hierarchische Struktur, die dem internen Datenfluss folgt. Datenprovider ohne Abhängigkeiten werden an den Anfang der Berechnungsreihenfolge gesetzt. Zusätzlich wird für jeden Datenprovider der benötigte Speicherplatz reserviert. Die Ergebnisse der Datenprovider werden in Tabellen mit flexibler Struktur abgelegt, wobei die Ausgaben der einzelnen Datenprovider den entsprechenden Spalten zugeordnet sind.

Der logische Datenfluss, der sich aus dem Netzwerk der Datenprovider ergibt (siehe Abbildung 7.3, links), wird in Form einer Baumstruktur dargestellt (siehe Abbildung 7.3, rechts). Die Abhängigkeiten werden am Beispiel der Datenquelle C verdeutlicht: Diese kann ihre Berechnungen erst ausführen, nachdem die Ausgaben der Datenquellen A und B bereitgestellt wurden, entsprechend ihrer internen Logik.

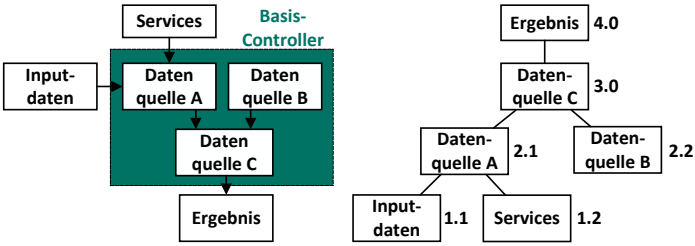

Abbildung 7.3: Erstellung des Datenflusses durch den Basis Controller während der Vorbereitungsphase. Die Zahlen im rechten Diagramm geben die Ausführungsreihenfolge der Datenprovider an, die sich aus deren Abhängigkeiten ergeben (siehe [SMS21]).

In der Evaluierungsphase werden die Datenprovider über eine festgelegte Anzahl von Iterationen wiederholt aufgerufen (siehe Abbildung 8.12). Bei jeder Iteration passen sich die Ausgabedaten der Datenprovider gemäß ihrer definierten Funktion an. Jeder Provider liefert dabei einen spezifischen Teil des Datensatzes (siehe Tabelle 8.3). Die berechneten Daten werden nach jeder Iteration zeilenweise in die entsprechenden Datentabellen eingefügt und abschließend in einer Gesamttabelle zusammengefasst. Dieses Verfahren ermöglicht die konsistente und zugleich flexible Erstellung von Datensätzen.

7.3 Methoden zur Erzeugung von Benutzeraktionen

Benutzeraktionen beziehen sich auf die Interaktionen, die ein Benutzer mit einem System durchführt. Diese Interaktionen können auf verschiedene Weisen erfolgen, wie z.B. über Berührung, Sprache, Gesten [SSM+22] oder Bedienelemente. Diese Aktionen sind im Kontext dieser Dissertation als spezifische Eingaben oder Verhaltensweisen definiert, die durch die Szenarien beeinflusst werden (siehe Definition 6.4). Mithilfe des entwickelten Tools CAGEN wird sowohl die Erzeugung von Kontext (siehe Definition 6.2) als auch die Generierung von Datensätzen zur Abbildung der Benutzeraktionen ermöglicht. Dies erlaubt eine gezielte Untersuchung und Analyse der Fahrerinteraktionen im Zusammenhang mit dem Kontexterzeugungssystem. Dadurch können Erkenntnisse über das Verhalten und die Präferenzen des Fahrers gewonnen und überprüft werden.

In dieser Dissertation liegt der Fokus auf Benutzeraktionen, die sich auf selbstlernende Komfortfunktionen beziehen. Im Automobilbereich sind dies Funktionen, die sich im Laufe der Zeit an die Präferenzen und Gewohnheiten des Fahrers anpassen, um den Fahrkomfort zu erhöhen. Solche Funktionen nutzen Technologien des maschinellen Lernens (siehe Abschnitt 2.1), um Daten über das Fahrverhalten, die Umgebung und die individuellen Vorlieben des Fahrers zu erfassen und daraus angepasste Einstellungen und Empfehlungen abzuleiten.

Im Folgenden werden beispielhafte selbstlernende Komfortfunktionen vorgestellt:

- **Selbstlernende Sitzposition**: Diese Funktion beobachtet die Sitzposition des Fahrers und justiert sie automatisch beim Betreten des Fahrzeugs.
- **Selbstlernende Klimatisierung**: Eine solche Funktion erkennt die bevorzugte Innentemperatur des Fahrers und passt sie automatisch an, sobald das Fahrzeug gestartet wird.
- **Selbstlernende Unterhaltungseinstellungen**: Hierbei handelt es sich um eine Funktion, welche die präferierten Einstellungen für Musik, Radiosender und andere Unterhaltungsmedien des Fahrers identifiziert und

diese automatisch einstellt, sobald das Fahrzeug in Betrieb genommen wird.

- **Selbstlernende Navigationsrouten**: Diese Funktion erkennt bevorzugt von einem Fahrer gewählte Strecken und schlägt diese automatisch vor, sobald das gewünschte Ziel eingegeben wird.

Diese Funktionen zeigen den zunehmenden Einsatz von maschinellem Lernen und künstlicher Intelligenz (siehe Kapitel 2) im Automobilbereich zur individuellen Optimierung des Fahrkomforts. Ihre Implementierung beruht auf der Erfassung von Daten, die von Sensoren wie GPS, Kameras und Beschleunigungssensoren bereitgestellt werden. Um selbstlernende Komfortfunktionen realitätsnah zu validieren, ist es notwendig, authentische Benutzerinteraktionen zu generieren. Der folgende Abschnitt untersucht verschiedene Ansätze zur Erzeugung solcher Benutzeraktionen. Dabei wurden drei konzeptionelle Ansätze analysiert (siehe Abbildung 7.4). Die Methode zur Generierung dieser Benutzeraktionen wird im Folgenden als „Benutzerlogik“ bezeichnet.

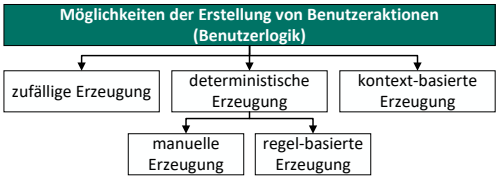

Abbildung 7.4: Darstellung der Optionen für die Erstellung von Benutzeraktionen. (siehe Abbildung 5.2, Bereich: Szenarioerzeugung, SE2)

Die erste untersuchte Methode zur Generierung von Benutzeraktionen erfolgt zufällig, ohne dabei auf logische Zusammenhänge mit den zuvor erstellten Kontextdaten einzugehen. Diese Herangehensweise ist technisch unkompliziert und ermöglicht die automatische Erzeugung von Testdaten. Allerdings überwiegen die Nachteile dieses Ansatzes. Durch die fehlende Abhängigkeit der Benutzeraktionen vom Kontext wird die Realität nicht ausreichend abgebildet, was zu unrealistischen Testszenarien führt und potenziell eine fehlerhafte Interpretation des Verhaltens des SuT begünstigt. Zudem ist diese Methode ungeeignet, gezielte Testdaten für spezifische Szenarien in bestimmten Anwendungsbereichen zu generieren. Darüber hinaus bietet die zufällige Erzeugung

keine Kontrolle über die Interaktionsmuster, wodurch die Aussagekraft und Verwendbarkeit der generierten Testdaten eingeschränkt werden.

Eine weitere Methode zur Erzeugung von Benutzeraktionen besteht in der Anwendung deterministischer Regeln. Hierbei wurden zwei Ansätze untersucht: die manuelle Erstellung durch Fachkräfte sowie die regelbasierte Erzeugung mittels vordefinierter Regeln. Bei der manuellen Erstellung berücksichtigt die Fachkraft spezifische Testziele, zu prüfende Funktionen und die zugrunde liegenden Szenarien. Auf dieser Basis werden gezielte Szenarien entwickelt, die relevante Anwendungsfälle abdecken. Die Fachkraft definiert dabei, welche Interaktionen zwischen dem Benutzer und dem System stattfinden sollen, wobei sowohl die Benutzereingaben als auch die erwarteten Systemreaktionen berücksichtigt werden. Die regelbasierte Erzeugung von Szenarien erfolgt durch die Anwendung vordefinierter Regeln, die das Systemverhalten in verschiedenen Situationen festlegen. Diese Regeln werden im Voraus von Entwicklern spezifiziert und definieren, wie das System auf bestimmte Eingaben oder Zustände reagiert. Ein Beispiel hierfür ist eine Regel, die festlegt, dass die Klimaanlage aktiviert wird, wenn die Innentemperatur einen festgelegten Schwellenwert überschreitet. Beide Ansätze ermöglichen die Generierung nachvollziehbarer Szenarien und sind daher geeignet, um spezifische Szenarien mit besonderen Randfällen zu testen. Für eine umfassende Validierung sind sie jedoch eingeschränkt nutzbar. Die „manuelle Erzeugung“ ist aufgrund des steigenden Programmieraufwands bei zunehmender Anzahl zu testender Szenarien ineffizient. Ebenso ist die „regelbasierte Erzeugung“ für eine Validierung ungeeignet, da sie lediglich vordefiniertes Verhalten prüft und keine dynamischen Benutzeraktionen abbilden kann.

Die dritte und bevorzugte Methode besteht in der Anwendung einer kontextbasierten Benutzerlogik, die sowohl zufällige als auch deterministische Elemente kombiniert. Dabei werden Wahrscheinlichkeiten verwendet, um das Auftreten von Benutzeraktionen in Abhängigkeit von spezifischen Merkmalen oder definierten Szenarien zu steuern, was zu einer Variation der erzeugten Interaktionen führt. Gleichzeitig werden deterministische Komponenten integriert, um eine gezielte Steuerung des Erzeugungsprozesses sicherzustellen.

Im folgenden Kapitel wird die kontextbasierte Benutzerlogik mithilfe der morphologischen Matrix methodisch (siehe Abbildung 6.4) beschrieben. Die morphologische Matrix dient als Instrument zur Verknüpfung der Merkmale und Parameter von Benutzeraktionen. Dadurch können unterschiedliche Szenarien

modelliert und die Benutzerlogik gezielt an spezifische Kontextbedingungen angepasst werden.

7.4 Kontextbasierte Benutzerlogik durch Verwendung der morphologischen Matrix

Die Anwendung der kontextbasierten Benutzerlogik wird durch die Notwendigkeit eines Zustandsübergangsmodells motiviert, das sowohl zufällige Eigenschaften integriert als auch nachvollziehbare Zusammenhänge ermöglicht. Zur Umsetzung dieser Benutzerlogik werden die Merkmale der morphologischen Matrix aus den Tabellen 6.1 und 6.2 herangezogen. Die Wahrscheinlichkeit des Auftretens einer Benutzeraktion wird in Abhängigkeit der Merkmale oder der in Abschnitt 6.2.1 definierten Wegzwecke festgelegt.

Im Nachfolgenden wird die Konzeption anhand eines Beispiels erläutert: Für ein Szenario mit den Merkmalen „Sommer“ und „Mittag“ sollen Benutzeraktionen zur Steuerung einer Sitzheizung erzeugt werden. In diesem spezifischen Szenario ist zu erwarten, dass ein Benutzer die Sitzheizung nicht verwendet und die Sitzheizung hauptsächlich im Zustand „aus“ verbleibt. Im Gegensatz dazu kann erwartet werden, dass die Sitzheizung im „Winter“ und am „Abend“ mit einer höheren Wahrscheinlichkeit vom Benutzer auf „hoch“ gestellt wird. Das Ziel der kontextbasierten Benutzerlogik besteht darin, das erwartete Verhalten zu beobachten und Testdaten zu erzeugen, die das Verhalten widerspiegeln. Zu diesem Zweck werden im folgenden die identifizierten Schritte zur Erzeugung von kontextbasierten Benutzeraktionen konzeptionell beschrieben. Die Implementierung dieser Schritte wird in Abschnitt 8.3 beschrieben, unter Verwendung einer definierten Wahrscheinlichkeitsmatrix (siehe Tabelle 8.4).

1. **Definition der Benutzeraktionen** Die Festlegung der möglichen diskreten Zustände definiert die verschiedenen Interaktionen, die durch einen Benutzer beeinflusst werden können. Dazu gehören Zustände wie „an“, „aus“, „niedrig“, „mittel“ und „hoch“.

2. **Definition der Wahrscheinlichkeitsmatrix** Die Erstellung einer Wahrscheinlichkeitsmatrix beinhaltet die Festlegung der Wahrscheinlichkeiten, mit denen bestimmte Zustände eintreten. Diese Matrix dient dazu,

die Eintrittswahrscheinlichkeiten für verschiedene Ereignisse oder Interaktionen im System zu modellieren.

3. **Berechnung der Wahrscheinlichkeit** Die Berechnung der Wahrscheinlichkeit für jeden Zustand erfolgt durch die Anwendung der definierten Wahrscheinlichkeitsmatrix in Kombination mit den ausgewählten Konfigurationen aus der morphologischen Matrix. Dadurch kann die kombinierte Wahrscheinlichkeit für das Auftreten jedes Zustands bestimmt werden. Beispielsweise erhöht die Auswahl der Ausprägung „Winter“ die Wahrscheinlichkeit, dass die Sitzheizung im Zustand „aus“ ist, während die Ausprägung „Sonne“ diese Wahrscheinlichkeit verringert.

4. **Erzeugung möglicher Ereignispunkte** Dieser Schritt umfasst die Generierung potenzieller Zeitpunkte und Orte, an denen Benutzeraktionen auftreten können. Dies kann bedeuten, dass Interaktionen zu festgelegten Zeitintervallen und innerhalb definierter geografischer Bereiche erzeugt werden.

5. **Zuweisung des Zustandes** Im abschließenden Schritt erfolgt die Auswahl des zuvor generierten Ereignispunkts unter Berücksichtigung der in Schritt 3 kombinierten Wahrscheinlichkeit der Zustände. Dadurch werden Benutzeraktionen generiert, die den vorgegebenen Wahrscheinlichkeiten entsprechend den ausgewählten Eigenschaften der morphologischen Matrix entsprechen.

Durch dieses Vorgehen wird die Grundlage geschaffen, eine Vielzahl von Szenarien zu generieren, bei denen die Benutzeraktionen kontextabhängig sind. Im Gegensatz zur manuellen Erstellung von Interaktionen durch einen Experten für jedes Szenario, ermöglicht diese Methode dem Experten, gezielt die Auftretenswahrscheinlichkeiten festzulegen, indem er die Werte in der Wahrscheinlichkeitsmatrix (siehe Schritt 2) definiert. Dadurch wird ein Kompromiss erreicht, der den Aufwand bei der Szenarioerstellung verringert und gleichzeitig ein nachvollziehbares Verhalten der Benutzeraktionen sicherstellt. Die kontextbasierte Erzeugung von Benutzeraktionen erfüllt damit die Anforderung AD4. Die Umsetzung dieses Vorgehens wird in Abschnitt 8.3 beschrieben, wobei eine definierte Wahrscheinlichkeitsmatrix verwendet wird (siehe Tabelle 8.4).

8 Prototypische Umsetzung von CAGEN im Kontext einer selbstlernenden Komfortfunktion

Auf Basis der konzeptionellen Beschreibung des Kontexts und dessen Aufteilung (siehe Kapitel 7) sowie der Erstellung von Benutzeraktionen (siehe Abschnitt 7.3) erfolgt die prototypische Umsetzung dieser Ansätze am Beispiel einer selbstlernenden Komfortfunktion. Das Tool CAGEN generiert aus der logischen Szenariodefinition operationalisierbare Testdaten. Durch diese Transformation wird der Übergang von der konzeptuellen Modellierung zur tatsächlichen Ausführung und die Validierung realisierbarer Szenarien ermöglicht. Die Modularität der Systemkomponenten trägt nicht nur zur Wartungs- und Erweiterungsfähigkeit der Plattform bei, sondern versetzt sie auch in die Lage, die Anforderungen der spezifischen Simulationsumgebung – insbesondere jene des SSIB – zu erfüllen. Abbildung 8.1 stellt die strukturierte Zusammensetzung der CAGEN-Softwarearchitektur dar und illustriert die Interaktion zwischen externen Datenquellen, spezifischen Services und dem übergeordneten Basis-Controller.

8.1 Services zur Datenbereitstellung in Simulationsprozessen

Die Services agieren als Lieferanten externer Daten, die für die Modellierung und Durchführung der Simulationsprozesse erforderlich sind. Das CAGEN-Modell umfasst dabei Services wie den Konfigurations-Service, den Positions-Service und den TableManager-Service. Diese Dienste gewährleisten die Versorgung der Provider mit initialen Daten, die für die Ausführung der Simulation notwendig sind. Alle Services werden nach dem Singleton-Entwurfsmuster realisiert, sodass von jeder Service-Klasse genau eine Instanz existiert.

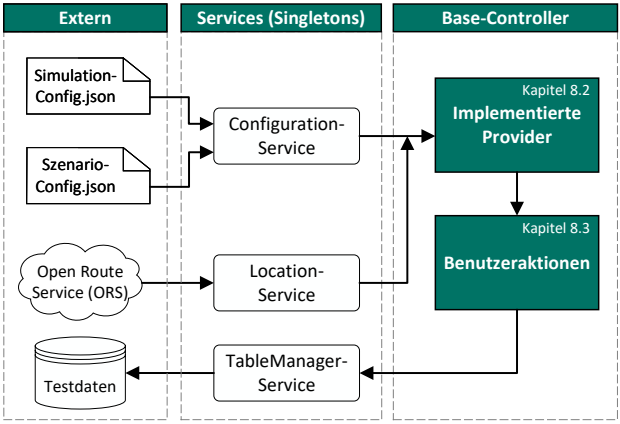

Abbildung 8.1: Übersicht der Komponenten und Schnittstellen von CAGEN (siehe Abbildung 5.2, Bereich: Szenarioerzeugung, SE1.1)

Definition 8.2 Singleton-Entwurfsmuster: Ein Singleton ist ein Entwurfsmuster (Design Pattern) in der Softwareentwicklung. Es stellt sicher, dass von einer bestimmten Klasse nur eine einzige Instanz existiert. Diese Instanz kann global abgerufen werden. Das Singleton-Muster kontrolliert so den Zugriff auf eine Ressource oder ein Objekt. Der Konstruktor ist privat, und die Klasse bietet eine statische Methode zum Zugriff auf die Instanz.

Diese singuläre Instanz sorgt dafür, dass die Konfigurationsinformationen über sämtliche Simulationsiterationen hinweg konstant bleiben. Im Gegensatz zu den Providern, die dynamische Werte aufweisen, bleiben die durch die Services bereitgestellten Daten unverändert, wodurch die Konsistenz der simulationsübergreifenden Daten garantiert wird. Die Implementierung als Singleton stellt zudem sicher, dass die Services einheitlich operieren, wodurch die Vorhersagbarkeit der Simulationsergebnisse optimiert wird. Diese Services bilden die Basis für weiterführende Komponenten des Systems, indem sie die notwendigen Rahmenbedingungen für einen akkuraten Ablauf der Simulation schaffen.

Konfiguration-Service: Zentrale Steuerung und Personalisierung der Simulationsparameter

Der Konfigurations-Service ist ein zentrales Element innerhalb der Service-Architektur des Systems. Dieser Service wird durch zwei JavaScript Object Notation (JSON)-Dateien initialisiert, welche die grundlegenden Einstellungen für die Simulation definieren. Die erste Datei, „Simulation-Config“, legt Parameter wie die Anzahl der Iterationen fest, die die Häufigkeit der während der Fahrt erfassten Datenpunkte bestimmen. Die zweite Konfigurationsdatei beinhaltet Informationen über die definierten Wegzwecke (siehe Unterabschnitt 6.2.1). Diese Datei ermöglicht es dem Konfigurations-Service, das System über die logischen Szenarien zu informieren. Ebenso werden in dieser Datei Informationen über den Fahrer konfiguriert, wodurch der Konfigurations-Service auch für die Personalisierung der Fahrsimulation anhand des Benutzerprofils des Fahrers verwendet wird.

Location-Service: Bereitstellung Positionsdaten für eine geografische Simulation

Der Location-Service nutzt eine API eines externen Anbieters, um Positionsdaten in Form von Koordinaten für definierte Start- und Endpunkte bereitzustellen. Diese Koordinaten, basierend auf Längen- und Breitengradangaben, werden durch die Vorgaben der logischen Szenarien determiniert. In jeder Simulationsiteration werden aktuelle Koordinaten abgerufen, die anschließend integraler Bestandteil der Simulationsergebnisse sind. Diese Koordinaten liefern die räumliche Basis für jede Iteration und dienen sowohl räumlichen Analysen als auch der Navigation innerhalb der simulierten Umgebung. Die vom Location-Service bereitgestellten Positionsdaten werden von weiteren Providern innerhalb des Systems genutzt, um beispielsweise Reisezeiten zu kalkulieren, Routen zu visualisieren oder Umgebungsvariablen zu analysieren. Die Integration des Open Route Services (ORS) stellt dem System eine Datenbasis zur Verfügung, ohne dass diese intern vorgehalten werden muss. Dies steigert die Genauigkeit der Simulation und ermöglicht eine anpassungsfähige Implementierung verschiedener geografischer Kontexte und Szenarien.

TableManager-Service: Strukturierte Speicherung und Verwaltung von Simulationsdaten

Der TableManager-Service ist verantwortlich für das Speichern und Organisieren aller von den Providern generierten Datenwerte in Tabellenform (siehe Tabelle 8.1). Dies schließt die Initialisierung der Tabellen mit ein, wobei der TableManager für jeden Provider entsprechende Strukturen erstellt. Diese Strukturen differenzieren zwischen Zeilen für Iterationsverläufe und Spalten für Kontextinformationen und umfassen unterschiedliche Kontextkategorien: den Realen-Kontext, den Sensor-Kontext sowie den Benutzer-Kontext. Ergänzend werden Tabellen für die Zustände von Objekten sowie für Werte von Hilfsklassen angelegt. Die Spalten der Tabellen sind durch eindeutige IDs gekennzeichnet, wie zum Beispiel „ID_Temperatur“ oder „ID_Location_Lat“ (siehe Tabelle 8.3). Während der Initialisierung festgelegte Tabellendimensionen werden zu Beginn mit dem Wert NaN initialisiert, welcher im weiteren Verlauf der Simulation durch gültige Daten ersetzt werden.

Tabelle 8.1: Erzeugter Beispieldatensatz unter Verwaltung des TableManager-Services. Die jeweiligen Spalten werden iterativ durch die Provider befüllt (siehe Abschnitt 8.2).

Iteration	Lat	Lon.	Pass.	Nied.	Temp	Zeit	Zustand
1	-0.00365307	0.00410699	1	0	2.5210	1628193600	1
2	49.01780443	8.40138728	1	0	22.0714	1628193609	1
3	49.00624158	8.40223244	1	0	23.9221	1628193617	0
...	...	...	...	...	...	...	...
98	48.69749912	9.00804276	2	0	34.4071	1628194438	1
99	48.69436782	8.99938095	2	0	37.1423	1628194446	1
100	48.70202912	8.99723847	2	0	32.3517	1628194455	0

8.2 Entwickelte Provider für die Testumgebung CAGEN

Die Provider sind verantwortlich für die Berechnung der Kontextwerte in jeder Iteration (siehe Tabelle 8.1). Jeder im Base-Controller aufgeführte Provider wird pro Iteration einmal ausgeführt, bevor die Berechnung der nächsten Iteration beginnt. Zunächst werden die Provider ohne Abhängigkeiten zu ande-

ren Providern ausgeführt. Diese Provider beziehen ihre Daten von externen Quellen über die entsprechenden Services. Anschließend werden die Provider ausgeführt, die von den zuvor ausgeführten Providern abhängen (siehe Abbildung 7.3). Die Provider sind für die Berechnung der Kontextwerte zu jeder Iteration zuständig. Jeder im Base-Controller aufgelistete Provider kommt in der Iteration einmal zum Einsatz, bevor es mit der Berechnung der nächsten Iteration weitergeht. Zunächst werden die Provider ohne Abhängigkeiten zu anderen Providern ausgeführt. Ihre Daten erhalten sie von den Services aus externen Quellen. Anschließend werden die Provider ausgeführt, die eine Abhängigkeit zu den zuerst ausgeführten Providern haben. Bei der Implementierung der Provider werden zunächst die Abhängigkeiten festgelegt, die die Reihenfolge und den Kontext bestimmen, für den die Werte berechnet werden (siehe Abbildung 8.2). Anschließend wird durch die Methode *setup()* Informationen aus den Konfigurationen bezogen, die für die Berechnung erforderlich sind. Mit dieser Funktion sollen Berechnungen für Werte durchgeführt werden, die über die gesamte Fahrt hinweg konstant bleiben.

Die Methode *get_default_data()* ordnet den initialen Wert der Tabelle zu Beginn der Iteration 0 zu. Danach kann über *calc_next_data()* der Wert für die aktuelle Iteration berechnet und zugeteilt werden. Für diese Berechnung kann unter anderem auf Werte vorheriger Iterationen zugegriffen werden. Dies gilt sowohl für die Werte der eigenen Tabelle als auch für die Werte anderer Kontexttabellen. Am Ende der Berechnung werden die Werte über den Data-Manager an der entsprechenden Stelle der Tabelle abgespeichert.

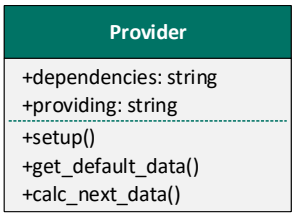

Abbildung 8.2: Basis Provider

Location-Provider

Die Klasse Location-Provider (siehe Abbildung 8.3), die von der Basisklasse Provider erbt, ist für die Verwaltung geografischer Standortinformationen und

die Bereitstellung von Navigationsdiensten innerhalb des Systems konzipiert, das Teil der actest.cagen-Bibliothek ist.

Beim Initialisieren erbt der Location-Provider grundlegende Eigenschaften durch den Aufruf des Konstruktors der Basisklasse. Anschließend definiert der Location-Provider seine Bereitstellungen (*providing*) und Abhängigkeiten (*dependencies*). Die Bereitstellungen umfassen Breiten- und Längengrade (*ID_Location_Lat, ID_Location_Long*) sowie die Entfernungen zum Start- und Zielort (*ID_Dist_to_Start, ID_Dist_to_Destination*). Der Location-Provider initialisiert eine Navigationsroute durch die Angabe von Start- und Zielkoordinaten. Die Variable *coordStart* stellt den Startpunkt einer Route mit Breiten- und Längengradangaben dar, während *coordEnd* den Endpunkt einer Route definiert. Durch den Einsatz des Open Route Service fordert die Klasse eine Routenberechnung basierend auf diesen Koordinaten an, wobei ein Entwicklungsmodus (*devMode*) für die Anpassung der Routenanfragen zur Verfügung steht. Dies illustriert die direkte Integration mit externen Routing-Diensten, die für die Berechnung realistischer Routen in Echtzeit genutzt werden. Der Open Route Service ist ein, frei zugängliches geografisches Informationssystem (GIS), das Routenplanungs- und Navigationsdienste anbietet. Entwickelt und unterstützt von der Geoinformatik-Abteilung der Universität Heidelberg, basiert der Dienst auf Daten von OpenStreetMap (OSM), der größten benutzergenerierten Weltkarte. Der Location-Provider ist eine zentrale Komponente innerhalb der CAGEN-Bibliothek, die nicht nur für die Bereitstellung präziser geografischer Daten zuständig ist, sondern auch die Integration mit externen Navigationsdiensten über den Location-Service (siehe Unterunterabschnitt 8.1) wie dem OpenRouteService ermöglicht.

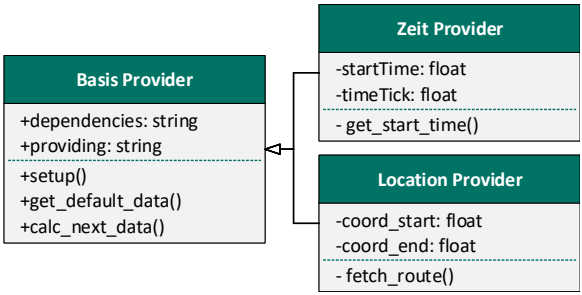

Abbildung 8.3: Location- und Zeit-Provider

Zeit-Provider

Die Aufgabenbereiche des Zeit-Providers (siehe Abbildung 8.3) erstrecken sich über die Initialisierung der Ausgangszeitpunkte und der Zeitintervalle, die Erzeugung zeitbezogener Daten für die einzelnen Durchläufe der Simulation sowie die Implementierung einer Schnittstelle für den Abruf zeitlicher Informationen in verschiedenen Darstellungsformaten.

Von essenzieller Bedeutung für die Funktionalität des Zeit-Provider ist die Akquisition von Standortdaten, die vom Location-Provider bereitgestellt werden und durch die Identifikatoren *ID_Location_Lat* sowie *ID_Location_Long* repräsentiert sind. Die vom Time-Provider generierten und zur Verfügung gestellten Daten umfassen *ID_Unix_Time*, welches die in der Simulation verwendete Zeit im UNIX-Format kodiert, und *ID_Readable_Time*, das ebendiese Zeitpunkte in einem für Menschen interpretierbaren Format darstellt.

Die Methode *get_start_time()* ist ein zentraler Bestandteil der Klasse. Sie ermöglicht die Festlegung einer Startzeit für die Simulation. Die Startzeit kann entweder zufällig innerhalb eines definierten Zeitraums gewählt oder basierend auf Informationen wie Jahreszeit und Tageszeit festgelegt werden. Dies ermöglicht eine flexible Anpassung der Simulation an spezifische Szenarien.

In der *setup()*-Methode wird die Dauer einer Route, die vom Location-Provider bereitgestellt wird, abgerufen und verwendet, um den Zeittakt (*timeTick*) der Simulation zu berechnen. Dieser Zeittakt bestimmt die Zeitspanne, die in der Simulation zwischen zwei Iterationen vergeht. Er basiert auf der Gesamtdauer der Route und der in der Konfiguration festgelegten Anzahl der Iterationen.

Die Methoden *get_default_data()* und *calc_next_data()* dienen dazu, die Zeitinformationen für den Start der Simulation bzw. für jede Iteration der Simulation zu generieren und in den entsprechenden Tabellen abzulegen. Diese Methoden gewährleisten, dass alle Komponenten des Systems Zugang zu konsistenten und genauen Zeitdaten haben.

Temperatur-Provider

Die Klasse Temperatur-Provider (siehe Abbildung 8.4) ist ein spezialisierter Datenprovider, der innerhalb der CAGEN-Bibliothek definiert wurde. Diese Klasse erbt von der übergeordneten Basisproviderklasse und ist dafür konzi-

piert, Temperaturdaten innerhalb eines bestimmten Bereichs zu generieren und bereitzustellen.

Die Temperaturwerte für jeden Monat sollen realistisch sein. Um dies zu erzielen, wurden für die Jahre 2018, 2019 und 2020 die aufgenommenen Wetterdaten jeden Monats aus dem *Monatlicher Klimastatus Deutschland* des Deutschen Wetterdienstes untersucht. Die Daten beinhalten Informationen bezüglich der Tiefsttemperatur, Durchschnittstemperatur und Höchsttemperatur an 120 verschiedenen Wetterstationen in Deutschland für den jeweiligen Monat.

Hierfür wurde aus der Menge aller aufgelisteten Durchschnittstemperaturen des Monats eine Normalverteilung mit Erwartungswert und Standardabweichung modelliert. Dabei entspricht der Durchschnitt aller Durchschnittstemperaturen dem Erwartungswert μ. Die Standardabweichung der Normalverteilung wird so festgelegt, dass der Durchschnitt von sowohl der Tiefsttemperaturen als auch der Höchsttemperaturen innerhalb $\pm 2\sigma$ entspricht. Somit werden 95,45% der möglichen Werte abgedeckt.

Das Vorgehen zur Bestimmung des Erwartungswertes und der Standardabweichung wird für jeden Monat durchgeführt. Somit ergibt sich für jeden Monat eine entsprechende Normalverteilung mit Erwartungswert und Standardabweichung in der folgenden Tabelle 8.2. Der Temperaturbereich für das logische Szenario ergibt sich für den jeweiligen Monat aus $\mu \pm 2\sigma$.

[°C]	Jan	Feb	Mär	Apr	Mai	Jun	Jul	Aug	Sep	Okt	Nov	Dez
μ	2,21	2,25	4,77	10,75	12,74	18,50	19,78	20,19	15,24	10,94	6,21	3,80
σ	4,13	5,01	5,68	6,67	6,18	5,93	6,34	6,25	6,26	5,59	5,18	4,23

Tabelle 8.2: Erwartungswert und Standardabweichung der Normalverteilungen der einzelnen Monate.

Der Provider ist darauf ausgerichtet, Daten von der *ID_Unixtime*, die vom Zeit-Provider bereitgestellt wird, abzurufen und bietet im Gegenzug die Werte *ID_Temperatur* an. Für die Ermittlung exakter Temperaturwerte sind die Attribute *minTemp* und *maxTemp* von zentraler Bedeutung, die jeweils die minimale und maximale Temperatur darstellen, die während des Simulationsverlaufs angenommen werden kann.

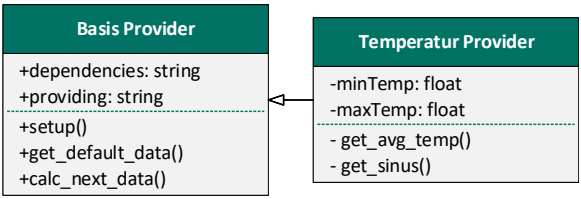

Abbildung 8.4: Temperatur-Provider

Die Kernfunktionalität des Temperatur-Providers wird durch die Methoden: *get_default_data* und *calc_next_data* realisiert:

- *get_default_data()* initialisiert die Temperaturdaten mit einem Standardwert in der ersten Zeile der zugehörigen Tabelle, der als vorläufiger Wert dient, bis die Berechnungen durchgeführt werden.
- *calc_next_data()* verwendet den Fortschrittsindikator der Iteration, um die Temperatur innerhalb des festgelegten Bereichs zu kalkulieren. Diese Methode nutzt einen linearen Interpolationsmechanismus, der die Temperaturwerte anhand des Fortschritts zwischen den Grenzwerten *minTemp* und *maxTemp* anpasst.

Perlin-Rausch-Provider

Die Aufgabe des Perlin-Rausch-Providers besteht darin, Signale durch einen Rauschwert zu überlagern.

Perlin-Rauschen ist eine Technik zur Erzeugung von natürlichen Mustern und Texturen, die 1983 von Ken Perlin [133] entwickelt wurde. Im Gegensatz zu weißem Rauschen, das gänzlich zufällige Werte erzeugt, bietet Perlin-Rauschen eine kontrollierte Zufälligkeit mit einer glatten, wellenförmigen Struktur. Es wird in der Computergrafik und Spielentwicklung verwendet, um realistische Texturen wie Feuer, Rauch, Wolken, Wasserwellen oder Terrainlandschaften zu simulieren [134]. Dabei wird zunächst eine diskrete Folge verwendet, die als Gitter bezeichnet wird. Die Glieder der Folge werden Raster- bzw. Gitterpunkte genannt und sind im gleichmäßigen Abstand zueinander entfernt. Der Wert jeder dieser Punkte ist eine Nullstelle der resultierenden Perlin-Funktion. Für jeden dieser Punkte wird ein reellwertiger, pseudozufälliger Gradientenwert

generiert, welcher zwischen -1.0 und +1.0 liegt. Die Werte für die kontinuierlichen Punkte zwischen den diskreten Gitterpunkten werden durch die Funktion *noise(x)* definiert und durch lokale Interpolation mithilfe einer Interpolationsfunktion bestimmt. Die Auswahl der Interpolationsfunktion hat eine Auswirkung auf den späteren Verlauf der Rauschfunktion. Die Gradienten an den diskreten Gitterpunkten können durch Tangenten dargestellt werden. Dabei zählt die übliche Geradengleichung zur Beschreibung der Tangenten.

$$h_i(x) = g_i \cdot (x - i) \tag{8.1}$$

Dabei beschreibt g_i den Gradientenwert der Geraden, die die x-Achse an der Position i schneidet. Mit *i* werden die jeweiligen diskreten x-Werte der Gitterpunkte verbunden. Über das Einheitsintervall werden die Endpunkte für die Interpolation bei i = 0 und i = 1 festgelegt. Dadurch ergibt sich aus der allgemeinen Tangentengleichung die zwei Gleichungen für aufeinanderfolgende Gitterpunkte. Dabei beschreiben g_0 und g_1 die jeweiligen Steigungen an den entsprechenden Gitterpunkten. Es ergeben sich die beiden Geradengleichungen für die ersten beiden Gitterpunkte:

$$\begin{aligned} h_0(x) &= g_0 \cdot x, \\ h_1(x) &= g_1 \cdot (x - 1) \end{aligned} \tag{8.2}$$

Anschließend wird in den Gleichungen x mit $\dot{x}$ substituiert, damit es sich auf das Einheitsintervall bezieht. Somit soll eine Unterscheidung zwischen dem Einheitsintervall und dem Gesamtintervall gemacht werden. Somit ergibt sich das Schaubild 8.5.

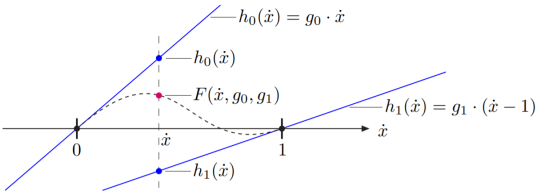

Abbildung 8.5: Interpolation der Tangentialgeraden im [0, 1]-Intervall. Abgewandelte Darstellung auf Basis von [133].

Zur Bestimmung der Interpolationsfunktion (vgl. englisch smootherstep function) F($\dot{x}$, g_0, g_1) wird die Verwendung einer s-förmige Überblendungsfunktion $s(x)$ vorgeschlagen.

$$s(x) = 10x^3 - 15x^4 + 6x^5 \tag{8.3}$$

Hierbei handelt es sich um ein Polynom 5. Grades, deren sowohl erste als auch zweite Ableitung an den Randpunkten des Einheitsintervalls gleich 0 sind. Dadurch entstehen C2-kontinuierliche Übergänge zwischen den benachbarten Segmenten. Somit ergibt sich die folgende Interpolationsfunktion 6. Grades:

$$\begin{aligned} F(\dot{x}, g_0, g_1) &= h_0(\dot{x}) + (10\dot{x}^3 - 15\dot{x}^4 + 6\dot{x}^5) \cdot (h_1(\dot{x}) - h_0(\dot{x})), \\ &= g_0 \cdot \dot{x} + (10\dot{x}^3 - 15\dot{x}^4 + 6\dot{x}^5) \cdot (g_1 \cdot (\dot{x} - 1) - g_0 \cdot \dot{x}) \end{aligned} \tag{8.4}$$

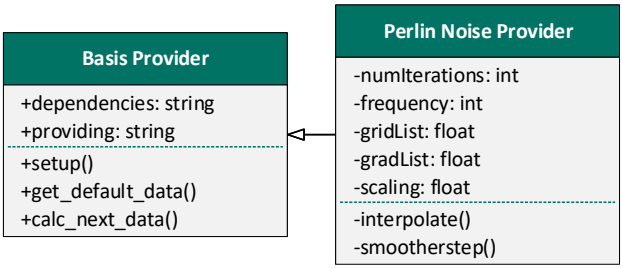

Abbildung 8.6: Perlin Rausch Provider

Der Perlin-Provider ist abhängig von dem Temperatur-Provider und wird nach diesem ausgeführt. Abhängig vom konfigurierten Wetter wird ein Perlin-Rauschen für die Berechnung erzeugt. Als Ergebnis werden Temperaturwerte für den Realen-Kontext gespeichert. Dieser befindet sich in der Spalte mit dem Attributnamen *ID_Temperatur* und überschreibt somit den ursprünglichen Wert aus dem Temperatur-Provider. *numIterations* beschreibt die Anzahl der Iterationen, die durch die Konfiguration bereitgestellt wird. Zudem wird mit *frequency* die Anzahl der gewünschten Gitterpunkte beschrieben. Die resultierenden Gitterpunkte werden als prozentualer Fortschritt des Durchlaufs

in *gridList* gespeichert. Jedem dieser Punkte wird mit *gradList* ein Gradientenwert zugeordnet. Ein Skalierungsfaktor wird mit *scaling* berücksichtigt.

Zunächst soll mit *setup()* das Gitter eingeführt werden. Das Gitter entspricht dabei dem Durchlauf der Fahrt. Entscheidend ist hierbei die maximale Zahl der Iterationen. Es wird die Menge an Gitterpunkten verwendet, die aus *frequency* hervorgeht. Diese Gitterpunkte bekommen in der entsprechenden Iteration den Wert 0 zugewiesen. Die Distanz zwischen zwei Gitterpunkten wird berechnet, indem die *numIterations* durch die *frequency* dividiert wird. Bei zehn Gitterpunkten auf hundert Iterationen, befinden sich somit die benachbarten Gitterpunkte im Abstand von zehn Iterationen zueinander. Den Gitterpunkten werden pseudozufällige Gradientenwerte zugeteilt. Diese Werte sollen zwischen -1.0 und +1.0 liegen. In der Methode *get_deafult_data()* wird ein zufälliger Gradientenwert für die 0. Iteration zugewiesen, da es sich dabei um den ersten Gitterpunkt handelt.

Anschließend wird mit *gridList* in jeder Iteration zunächst geprüft, ob es sich dabei um einen Gitterpunkt handelt oder nicht. Bei einem Gitterpunkt wird für den entsprechenden Eintrag in der Tabelle der Wert 0 eingetragen. Sollte es sich bei der Iteration nicht um einen Gitterpunkt handeln, so kann der Wert über die Interpolationsfunktion berechnet werden, indem die implementierte Methode *interpolate()* angewendet wird. Diese Funktion benötigt als Parameter den Gradienten des vorherigen und des nachfolgenden Gitterpunktes im aktuellen Intervallbereich. Diese werden durch *gradList* übergeben. Das Intervall wird im nächsten Schritt als Einheitsintervall mit einem Bereich von 0 bis 1 dargestellt, wobei die aktuelle Iteration einen prozentualen Wert zwischen den Grenzen darstellen soll. Anschließend werden mithilfe der Überblendungsfunktion *smootherstep()* die Tangenten beider Gitterpunkte miteinander interpoliert, sodass der Perlin-Wert für die aktuelle Iteration berechnet wird. Am Ende der Berechnung lässt sich der Wert noch mit *scaling* beliebig skalieren. Die resultierende Rauschkurve wird im Anschluss der Temperatur aus dem Temperatur-Provider dazu addiert.

Passagier-Provider

Der Passagier-Provider (siehe Abbildung 8.7) simuliert die Anzahl der Person einer Fahrt. Durch den Provider kann zusätzlich auch das zusteigen und aussteigen während einer Fahrt simuliert werden.

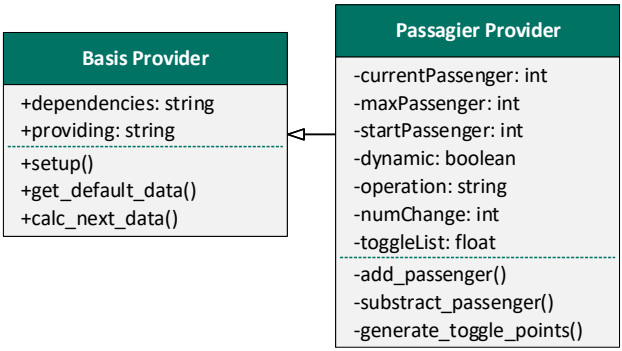

Abbildung 8.7: Passagier Provider

Die Berechnung der Passagierbesetzung hängt von keinem weiteren Provider ab. Diese Werte werden direkt der Tabelle des Sensorkontexts unter der Spalte mit dem Attributnamen *ID_PASSENGER* übergeben. Die Anzahl der Passagiere orientiert sich an die Konfiguration der Szenarien. Dabei beschreibt *currentPassenger* die aktuelle Anzahl der Passagiere. *maxPassenger* beschreibt die maximale Anzahl der Passagiere über die gesamte Fahrt. Diese Anzahl soll während des Durchgangs mindestens einmal erreicht werden. Über das Attribut *startPassenger* wird die Anzahl der Passagiere zum Start beschrieben. Mit *dynamic* wird die Entscheidung darüber getroffen, ob es sich beim Passagierverhalten um einen statischen oder dynamischen Verlauf handelt.

Bei einem statischen Verhalten bleibt die Passagierzahl über die gesamte Fahrt über konstant und die Passagieranzahl beim Start entspricht in jeder Iteration gleich der konfigurierten maximalen Anzahl an Passagieren. Bei einer dynamischen Fahrt hingegen wird sich die Anzahl der Passagiere an bestimmten Umschaltpunkten verändern. Bei dem Attribut *operation* wird festgelegt, ob eine Addition oder Subtraktion zur Berechnung des nächsten Wertes bei einem dynamischen Verhalten angewandt wird. Eine maximale Anzahl von zwei Zustandsänderungen der Passagiere wurde bei einem dynamischen Verhalten über das Attribut *numChange* festgelegt. Die konkreten Umschaltpunkte werden durch die Liste *toggleList* beschrieben.

Zunächst soll mit *setup()* eine Liste mit den Umschaltpunkten für den dynamischen Fall generiert werden. An diesen Punkten soll sich die Anzahl der

Passagiere verändern. Hierfür wird einerseits die Anzahl der Zustandsänderungen bestimmt und andererseits die Zeitpunkte der Veränderung. Die Anzahl der Änderungen wird zufällig zwischen 1 und 2 entschieden. Zur Bestimmung der Zeitpunkte soll die Fahrt als ein Wertebereich zwischen 0.0 und 1.0 dargestellt werden. Hierbei entspricht 0.0 dem Start und 1.0 dem Ziel des Durchlaufs. Abhängig von der Anzahl der Punkte wird ihnen ein zufälliger Float-Wert innerhalb dieses Wertebereichs zugeordnet. Dieser Wert stellt einen prozentualen Anteil der zu fahrenden Strecke dar. Bei mehreren Umschaltpunkten sollen diese einen Mindestabstand zueinander haben. Dies dient dazu, um die Realität besser abzubilden. Dieser Mindestabstand wurde zunächst auf 5% der Gesamtstrecke festgelegt. Sollte ein generierter Punkt sich innerhalb des definierten Mindestabstandes befinden, so wird der generierte Punkt verworfen und stattdessen ein neuer generiert. Dieser Vorgang erfolgt so lange, bis alle Punkte mit dem Mindestabstand zueinander liegen. Am Ende ergibt sich eine nach dem Fortschritt sortierte Liste *toggleList* mit verschiedenen Umschaltpunkten, die sich in einem Mindestabstand zueinander befinden. Die Generierung der Umschaltpunkte erfolgt mit der Methode *generate_toggle_points()*.

Bei der Methode *calc_next_data()* wird unterschieden zwischen einem statischen und einem dynamischen Verlauf. Sollte das Verhalten der Fahrt auf statisch konfiguriert worden sein, so ändert sich die Anzahl der Passagiere über die gesamte Fahrt nicht. Bei einer dynamischen Fahrt hingegen wird sich die aktuelle Anzahl im Laufe der Fahrt verändern. Hierzu wird die zuvor generierte Liste *toggleList* mit Umschaltpunkten aufgegriffen. Zur Berechnung des nächsten Wertes wird die aktuelle Iteration als ein prozentualer Wert der Gesamtiteration beschrieben. Dieser Wert wird sich im Laufe der Fahrt mit steigender Iteration erhöhen. Sobald dieser Wert einen zuvor definierten Umschaltpunkt übersteigt, wird abhängig von *operation* entweder *add_passenger()* oder *subtract_passenger()* ausgeführt und somit eine Veränderung des Passagierzustandes herbeigeführt.

Bei der Operation Addition ist *startPassenger* zu Beginn niedriger als *maxPassenger*. Somit werden weitere Passagiere hinzugefügt, um die maximale Anzahl im Verlauf der Fahrt zu erreichen. Dabei soll zwischen der Anzahl an Umschaltpunkten unterschieden werden. Ergibt sich durch die Konfiguration nur ein Umschaltpunkt, so wird nach Erreichen des Umschaltpunktes der Wagen vollständig auf die maximale Anzahl an Passagiere aufgefüllt. Ergeben sich jedoch mehre Umschaltpunkte, so wird bei jedem Punkt eine zufällige Anzahl

an Passagieren hinzugefügt. Diese Zahl liegt zwischen 1 und *maxPassenger* - *currentPassenger*. Sollte die maximale Anzahl vor dem letzten Umschaltpunkt bisher nicht erreicht worden sein, so wird diese beim Erreichen des Punktes komplett aufgefüllt. Wird hingegen die Subtraktion konfiguriert, so ist *start-Passenger* am Anfang gleich der *maxPassenger*. Demzufolge wird sich die Anzahl der Passagiere beim nächsten Umschaltpunkt verringern. Die Anzahl an Passagiere ist eine zufällige, ganzzahlige Zahl zwischen 1 und *currentPassenger*-1, damit sich mindestens noch eine Person im Wagen befindet.

Interferenz Provider

Der Interferenz-Provider (siehe Abbildung 8.8) ist für die Simulation der Störungen während der Fahrt zuständig. Bei den vereinzelten Störungen handelt es sich um die Ausprägungen: *Tunnel*, *Stau*, *Ampeln* (siehe Abbildung 6.4). Je nach Konfiguration wird eine der drei Störungen durch den Provider simuliert. Hierbei soll der Wert 1.0 ausgegeben, wenn sich das Fahrzeug gerade in einer der drei Störungen befindet. Sollte es sich jedoch nicht in einer Störung befinden, dann wird der Wert 0.0 ausgegeben.

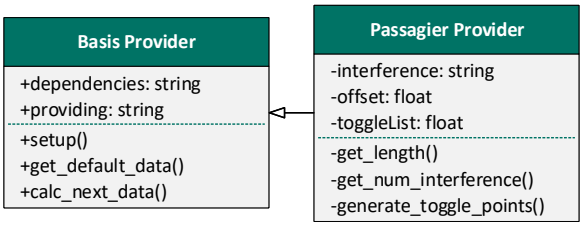

Abbildung 8.8: Interferenz Provider

Der Interferenz-Provider ist unabhängig von anderen Providern und somit nicht auf ihre Werte angewiesen. Der berechnete Wert wird in unter *ID_Interference* gespeichert. Das Attribut *interference* beschreibt die vom Konfiguration-Service konfigurierte Störung. *offset* beschreibt eine Mindestdistanz, die eine Störung zum Start- und Endpunkt haben soll. Mit der *toggleList* werden die Umschaltpunkte der auftauchenden Störung beschrieben.

Das Auftreten der Störung verläuft ähnlich zu der Liste an Umschaltpunkten beim Passagier-Provider. Die Punkte treten dabei abhängig des prozentualen Fortschritts des Durchlaufs auf. Hierzu soll mit *generate_toggle_points()* eine

toggleList mit den auftretenden Störungen erzeugt werden. Dies erfolgt zufällig, mithilfe einer Zahl zwischen 0.0 + *offset* und 1.0 - *offset*. Mit dem *offset* soll erreicht werden, dass keine Störung direkt zu Beginn oder am Ende der Fahrt auftritt. Der Wert der Verschiebung lässt sich beliebig einstellen. Die Anzahl der einzelnen Störungen wurde bei der Bestimmung der logischen Szenarien bereits als möglicher Wertebereich vorgegeben. Der Einfluss des Streckentyps wurde bei der Auswahl der entsprechenden Wertebereiche berücksichtigt. Um die konkrete Anzahl zu erhalten, wird aus dem Wertebereich ein zufälliger, ganzzahliger Wert mit *get_num_interference()* bestimmt.

Im Gegensatz zu dem Passagier-Provider soll *toggleList* sowohl den Start- als auch den Endpunkt der Störungen beinhalten. Hierzu soll ein prozentualer Wert, der die Länge darstellen soll, dem Startpunkt mit *get_length()* hinzuaddiert werden. Dabei unterscheiden sich die Längen der einzelnen Störungen voneinander. Die Anzahl der Iterationen pro Durchlauf wirkt sich ebenfalls auf die Berechnung aus. Ist die Anzahl der Stichproben im Vergleich zur Gesamtlänge hoch, so werden mehr Stichproben genommen und die entsprechenden Störungen existieren öfter in verschiedenen Iterationen. Ist sie im Vergleich gering, so werden die Störungen nur bei vereinzelten Iterationen angezeigt. Zusätzlich soll bei den Störungen ein Mindestabstand zwischen aufeinanderfolgenden Störungen eingehalten werden. Dieser Abstand variiert je nach Art der Störungen und wird mit *get_length()* berechnet.

Tunnel: Für den Fall *Tunnel* kann die Länge variieren. Die Mindestlänge in der Simulation soll daher bis zu 5% der Gesamtstrecke betragen. Bezüglich des Mindestabstands von zwei aufeinanderfolgenden Tunneln, soll dieser bei mindestens 20% liegen.

Stau: Der *Stau* ist die längste der drei Störungen und kann in der Simulation bis zu 20% der Gesamtstrecke ausmachen. Ebenso wie bei den Tunneln sollen zwei aufeinanderfolgende Staus mit mindestens 20% der Gesamtstrecke auseinanderliegen.

Ampel: Die *Ampel* soll als kürzeste der drei Störungen implementiert werden. Dieser Länge beträgt dabei bis zu 3% der Gesamtstrecke. Aufeinanderfol-

gende Ampeln sollen jedoch die einen kürzeren Abstand zueinander aufweisen. Im konkreten Fall beträgt dieser 5%.

Es ist zudem zu beachten, dass sich die Position der Störungen für den Fall *Tunnel* und *Ampeln* für aufeinanderfolgende Durchläufe nicht unterscheiden sollen. Bei der Störung *Stau* hingegen, kann das Auftreten der Störung variieren. Zu Beginn wird mit *get_default_data()* für alle drei Störungen der Wert 0.0 ausgegeben, da unter Berücksichtigung des *Offsets* keine Störung auftreten soll. Für jede folgende Iteration wird der Wert auf 0.0 gesetzt, es sei denn, das Auftreten der entsprechenden Störung ändert den Wert auf 1.0. Dies geschieht, wenn der prozentuale Fortschritt der Fahrt den Startwert der Störung überschreitet. In diesem Fall erfolgt eine Zustandsänderung auf 1.0. Dieser Zustand bleibt bestehen, bis der Fortschritt den Endpunkt der Störung überschritten hat.

Tabelle 8.3 fasst die implementierten Provider zusammen und fasst die Abhängigkeiten und bereitgestellten Signale zusammen.

Tabelle 8.3: Übersicht der Provider, der Abhängigkeiten und bereitgestellten IDs

Name Provider	Abhängig von	Bereitstellung von
Location	Open-Route Service	ID_Location_Lat/Long, ID_Speed
Zeit	Konfiguration	ID_Zeit
Temperatur	Location	ID_Temperatur
Perlin-Rauschen	alle	jede ID mit Rauschen
Passagier	Konfiguration	ID_Passenger
Interferenz	Location	ID_Interferenz

In Abbildung 8.9 ist ein Datensatz eines Szenarios unter Verwendung der Provider (siehe Tabelle 8.3) abgebildet.

8.3 Erzeugung von Benutzeraktionen

Neben der Simulation von Kontextdaten durch die implementierten Provider (siehe Kapitel 8) werden durch CAGEN davon abhängige Benutzeraktionen erzeugt. Diese Aktionen werden sowohl als „Aktionen“ wie auch auf Eng-

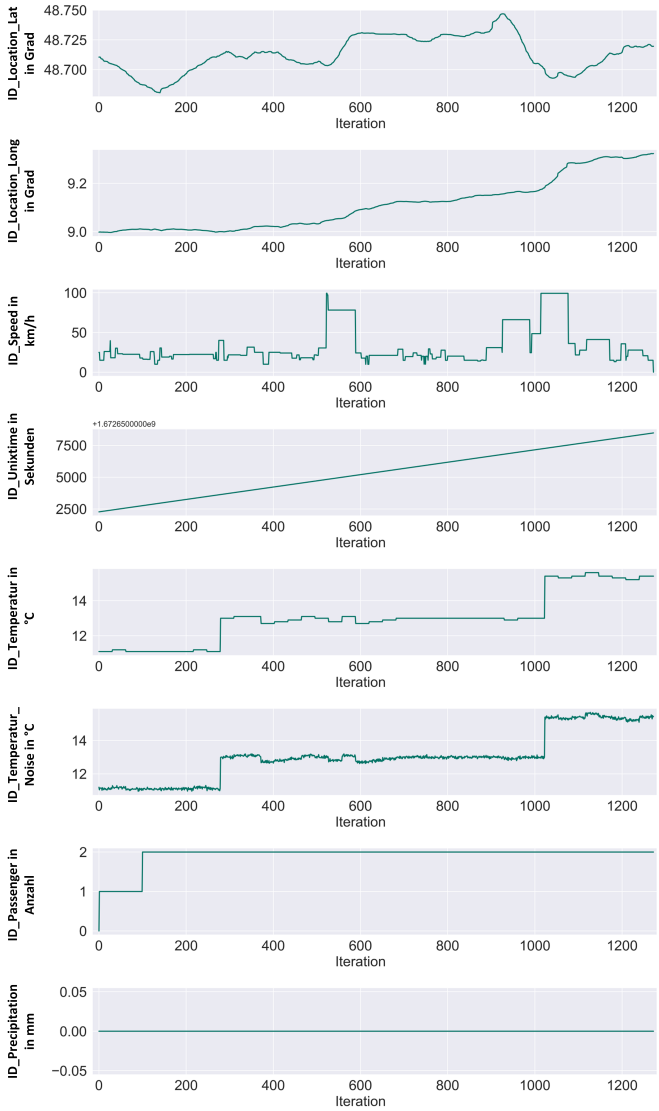

Abbildung 8.9: Darstellung der erzeugten Kontextdaten durch die vorgestellten Provider für Wegzweck „Arbeit“ (siehe Abbildung 5.2, Bereich: Szenarioerzeugung, SE3)

lisch als „Events“ bezeichnet. Die Aufgabe des implementierten SuT ist diese Aktionen zu erlernen und die Komfortfunktionen zu automatisieren (siehe Abschnitt 8.4). Im Folgenden werden die in Abschnitt 7.4 beschriebenen Schritte für konkrete Events umgesetzt.

Schritt 1: Definition der Benutzeraktionen

Die in dieser Dissertation betrachteten und modellierten Komfortfunktionen umfassen die Sitzmassage, die Sitzheizung und -belüftung und die Fenstersteuerung. Diese Funktionen können je nach Typ verschiedene Zustände annehmen, wobei jeder Zustand durch spezifische numerische Werte repräsentiert wird. Für die Funktionen der Sitzmassage und der Sitzheizung/-belüftung indiziert der numerische Wert 0 den Zustand „Aus“.

Im Gegensatz dazu symbolisiert bei der Fenstersteuerung und dem Schiebedach der Wert 0 den Zustand „offen“ und der Wert 1 den Zustand „geschlossen“. Weiterhin repräsentieren die Zahlen 1 bis 3 bei der Sitzmassage und der Sitzheizung/-belüftung verschiedene Intensitätsstufen. Zum Beispiel entspricht bei der Sitzmassage die Stufe 1 einer niedrigen Massageintensität, während bei der Sitzheizung die Stufe 3 die höchste einstellbare Heizstufe darstellt. Diese Einstellungen sind spezifisch für jede betrachtete Komfortfunktion und müssen je nach Bedarf angepasst werden. Die Benutzeraktionen (siehe Abschnitt 7.3) sind kontextbasiert, das bedeutet sie werden nicht zufällig erzeugt sondern basieren auf den gewählten Merkmalen der Kontextmerkmale (siehe Tabelle 6.1) und Benutzermerkmale (siehe Tabelle 6.2). Zu diesem Zweck werden in Tabelle 8.4 die Wahrscheinlichkeiten von Zuständen der Benutzeraktionen in Abhängigkeit beider Merkmale definiert.

Tabelle 8.4: Manuell festgelegte Wahrscheinlichkeiten der Zustände von Benutzeraktionen in Abhängigkeit von Merkmalen. (siehe Abbildung 5.2, Bereich: Szenarioerzeugung, SE2.1)

	Benutzeraktion	**Sitzmassage**				**Sitzheizung**				**Sitzbelüftung**				**Fenster**	
	Zustand	**0**	**1**	**2**	**3**	**0**	**1**	**2**	**3**	**0**	**1**	**2**	**3**	**0**	**1**
Jahreszeit	**Frühling**					0,6	0,3	0,1	0,0	0,8	0,1	0,1	0,0	0,6	0,4
	Sommer					0,7	0,2	0,1	0,0	0,0	0,3	0,4	0,3	0,8	0,2
	Herbst					0,4	0,2	0,2	0,2	0,7	0,2	0,1	0,0	0,3	0,7
	Winter					0,0	0,2	0,4	0,4	1,0	0,0	0,0	0,0	0,1	0,9
Tageszeit	**Morgens**					0,4	0,3	0,2	0,1	0,7	0,1	0,1	0,1	0,2	0,8
	Vormittags					0,7	0,1	0,1	0,1	0,0	0,3	0,4	0,3	0,8	0,2
	Mittags					0,5	0,4	0,1	0,0	0,4	0,2	0,2	0,2	0,7	0,3
	Nachmittags					0,7	0,1	0,1	0,1	0,6	0,2	0,1	0,1	0,6	0,4
	Abends					0,4	0,3	0,2	0,1	0,7	0,1	0,1	0,1	0,3	0,7
	Nachts					0,4	0,3	0,2	0,1	0,7	0,1	0,1	0,1	0,1	0,9
Wetter-konditionen	**Sonne**					0,7	0,2	0,1	0,0	0,3	0,3	0,2	0,2	0,6	0,4
	Wolken					0,5	0,2	0,2	0,1	0,6	0,2	0,1	0,1	0,2	0,8
	Regen					0,6	0,2	0,1	0,1	0,6	0,2	0,1	0,1	0,1	0,9
	Schnee					0,1	0,2	0,3	0,4	0,9	0,1	0,0	0,0	0,1	0,9
Störungen	**Stau**	0,0	0,0	0,5	0,5									0,1	0,9
	Tunnel													0,1	0,9
	Ampel													0,1	0,9
	Keine														
Streckentyp	**Autobahn**	0,0	0,1	0,3	0,6									0,1	0,9
	Überland													0,5	0,5
	Stadt													0,5	0,5
Fahrerprofil	**Standard**													0,6	0,4
	Frischluft-liebhaber													0,9	0,1
	Raucher													0,8	0,2
	Allergiker													0,0	1,0
	Rentner	0,0	0,2	0,6	0,2									0,3	0,7

Auf der linken Seite der Tabelle sind die Merkmale und die Ausprägungen dieser dargestellt (siehe Abschnitt 6.2). Die Ausprägungen der Merkmale werden über den Konfigurationsservice (siehe Unterunterabschnitt 8.1) an CAGEN weitergegeben.

Die Auswahl der Merkmale hat Auswirkung auf die Art und die Wahrscheinlichkeit der Benutzeraktionen. Jeder Zahlenwert symbolisiert die Wahrscheinlichkeit einer Aktivierung des entsprechenden Zustands der Komfortfunktion in Abhängigkeit des Merkmals. Beispielsweise wird die Komfortfunktion Fenster mit einer Wahrscheinlichkeit von 0,1 (10%) im Winter den Zustand 0 („offen“) besitzen und mit einer Wahrscheinlichkeit von 0,9 (90%) im Winter den Zustand 1 („geschlossen“). Aufbauend auf der Tabelle 8.4 werden im Weiteren die Erstellung von Benutzeraktionen anhand der Komfortfunktion Sitzheizung und den zwei Merkmalen „Jahreszeit“ und „Tageszeit“ beschrieben. Der Algorithmus führt diese Berechnungen für alle Merkmale für jede Komfortfunktion durch.

Schritt 2: Definition der Wahrscheinlichkeitsmatrix

In diesem Schritt wird die Wahrscheinlichkeitsmatrix für jeden Eingabeparameter manuell definiert. Jede Matrix ist durch die Dimensionen (*mxn*) gekennzeichnet, wobei *m* die Anzahl der Ausprägungen pro Merkmal und *n* die Anzahl der möglichen Zustände darstellt. Beispielsweise besitzt die Eingabe „Jahreszeit“ vier Merkmale *m* (Frühling, Sommer, Herbst, Winter) und vier mögliche Sitzheizungszustände *n*. Jede Zeile der Matrix repräsentiert eine diskrete Wahrscheinlichkeitsverteilung für einen Zustand, abhängig von einer bestimmten Eingabe. Die Summe der Werte in jeder Zeile muss eins ergeben, was die Gesamtwahrscheinlichkeit darstellt.

Für die Option „Sommer“ wird die Verteilung entsprechend der Tabelle 8.4 manuell definiert als „Sitzheizung Stufe 0“: 0,6, „Sitzheizung Stufe 1“: 0.3, „Sitzheizung Stufe 2“: 0,1 und „Sitzheizung Stufe 3“: 0,0. Zusätzlich dazu können Merkmale mit einem Gewichtungsfaktor *w* versehen werden. Diese Art der Modellierung erlaubt es, die Einflussstärke eines Merkmals zu steuern. Im genannten Beispiel haben alle Eingaben das Gewicht 1, was bedeutet, dass jede Eingabe gleich berücksichtigt wird. Abhängig vom Szenario könnte jedoch ein Eingabetyp, wie die „Jahreszeit“, eine stärkere Gewichtung erhalten als ein anderer, beispielsweise die „Tageszeit“.

Schritt 3: Berechnung der kombinierten Wahrscheinlichkeiten für jeden Zustand

Der nächste Schritt im Prozess besteht darin, die kombinierten Wahrscheinlichkeiten für einen bestimmten Satz von Merkmalen zu berechnen. Jedes Merkmal beeinflusst den Endzustand durch einen Wahrscheinlichkeitswert und ein zugeordnetes Gewicht. Auf dieser Grundlage lässt sich der kombinierte Wahrscheinlichkeitswert für jeden möglichen Zustand ermitteln. Die gewählten Merkmale werden durch den Konfigurationsservice (8.1) spezifiziert.

Im Folgenden wird das Vorgehen für die Benutzeraktion Sitzheizung anhand zweier Ausprägungen beschrieben, „Sommer“ des Merkmals „Jahreszeit“ und „Mittags“ des Merkmals „Tageszeit“. Aus den ausgewählten Merkmalen wird die Wahrscheinlichkeitsverteilung für den Zustand berechnet. Für jedes Merkmal wird ein Gewicht von eins definiert ($w_{Sommer}, w_{Mittag} = 1$). In diesem Beispiel besteht der Zustand aus vier möglichen Werten. Die Wahrscheinlichkeit jedes möglichen Zustands wird durch die ausgewählten Eingaben definiert:

- **Eingabe 1:** Jahreszeit = „Sommer“, Gewichtung $w_{Sommer} = 1$, Wahrscheinlichkeit $p = [0.7, 0.2, 0.1, 0.0]$
- **Eingabe 2:** Tageszeit = „Mittags“, Gewichtung $w_{Mittags} = 1$, Wahrscheinlichkeit $p = [0.5, 0.4, 0.1, 0.0]$

Um beide Wahrscheinlichkeiten der Eingaben und deren Gewichte zu kombinieren, werden die folgenden Berechnungen durchgeführt:

- **Wahrscheinlichkeit des Zustands 0:**

$$\mathcal{P}(x_{Sommer}) = w_{Sommer} \cdot p_{Zustand0} = 1 \cdot 0.7 = 0.7$$
$$\mathcal{P}(x_{Mittag}) = w_{Mittag} \cdot p_{Zustand0} = 1 \cdot 0.5 = 0.5$$
$$\mathcal{P}_{Zustand0} = \frac{\mathcal{P}(x_{Sommer}) + \mathcal{P}(x_{Mittag})}{w_{Sommer} + w_{Mittag}} = \mathbf{0.6}$$

- **Wahrscheinlichkeit des Zustands 1:**

$$\mathcal{P}(x_{Sommer}) = w_{Sommer} \cdot p_{Zustand1} = 1 \cdot 0.2 = 0.2$$
$$\mathcal{P}(x_{Mittag}) = w_{Mittag} \cdot p_{Zustand1} = 1 \cdot 0.4 = 0.4$$
$$\mathcal{P}_{Zustand1} = \frac{\mathcal{P}(x_{Sommer}) + \mathcal{P}(x_{Mittag})}{w_{Sommer} + w_{Mittag}} = \mathbf{0.3}$$

- **Wahrscheinlichkeit des Zustands 2:**

$$\mathcal{P}(x_{Sommer}) = w_{Sommer} \cdot p_{Zustand2} = 1 \cdot 0.1 = 0.1$$
$$\mathcal{P}(x_{Mittag}) = w_{Mittag} \cdot p_{Zustand2} = 1 \cdot 0.1 = 0.1$$
$$\mathcal{P}_{Zustand2} = \frac{\mathcal{P}(x_{Sommer}) + \mathcal{P}(x_{Mittag})}{w_{Sommer} + w_{Mittag}} = \mathbf{0.1}$$

- **Wahrscheinlichkeit des Zustands 3:**

$$\mathcal{P}(x_{Sommer}) = w_{Sommer} \cdot p_{Zustand3} = 1 \cdot 0.0 = 0.0$$
$$\mathcal{P}(x_{Mittag}) = w_{Mittag} \cdot p_{Zustand3} = 1 \cdot 0.0 = 0.0$$
$$\mathcal{P}_{Zustand3} = \frac{\mathcal{P}(x_{Sommer}) + \mathcal{P}(x_{Mittag})}{w_{Sommer} + w_{Mittag}} = \mathbf{0.0}$$

Die Summe jedes $P_{Zustand}$ muss 1 ergeben und entspricht der Wahrscheinlichkeitsverteilung der Zustände. Somit ergibt sich eine kombinierte Wahrscheinlichkeitsverteilung von $[0.6, 0.3, 0.1, 0.0]$. Die Verteilung lässt sich wie folgt interpretieren, die Sitzheizung wird zur Jahreszeit „Sommer“ und zur Tageszeit „Mittags“ zu 60% im Zustand 0 sein, zu 30% in Zustand 1, zu 10% in Zustand 2 und zu 0% in Zustand 3.

Schritt 4: Erzeugung möglicher Ereignispunkte und Intervalle

Nach der Berechnung der Wahrscheinlichkeiten für jeden Zustand ist das Ziel der folgenden Schritte, jeder einzelnen Iteration innerhalb einer vorgegebenen Gesamtzahl von Iterationen einen der möglichen Zustände zuzuweisen. Eine Iteration entspricht einem Datenpunkt des Location-Providers. Dabei soll die

Häufigkeit des Auftretens jedes Zustandes der zuvor berechneten Wahrscheinlichkeitsverteilung entsprechen.

Als Beispiel kann ein Szenario mit der Gesamtzahl der Iterationen $N = 100$ betrachtet werden. Die Wahrscheinlichkeitsverteilung für jeden Zustand der Sitzheizung ist $[0.6, 0.3, 0.1, 0.0]$. Die erwartete Ausgabe nach der Zuweisung eines Zustands zu jeder Iteration ist eine Liste mit der Länge 100: $[0, 0, 1, 1, 2, 0, ...]$. Die Ausgabeliste enthält 60x den Zustand 0, 30x den Zustand 1, 10x den Zustand 2 und 0x den Zustand 3. Die generierte Ausgabeliste entspricht dem simulierten Verlauf des Zustands. Jede Änderung in zwei aufeinanderfolgenden Elementen impliziert ein Ereignis, wie das Umschalten der Sitzheizung von Zustand 0 auf 1. Um sicherzustellen, dass die erzeugten Benutzeraktionen denen einer realen Person entsprechen, sollten diese nicht bei jeder Iteration geändert werden. Stattdessen ist es notwendig, dass jeder Zustand über mehrere Iterationen hinweg konstant bleibt, um ein realistisches Verhalten abzubilden.

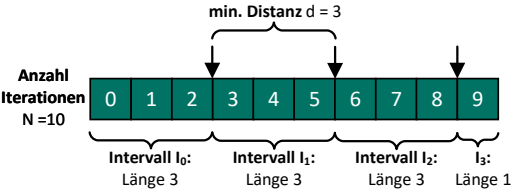

Abbildung 8.10: Darstellung des Mindestabstands $d_{min} = 3$ für $N = 10$

Um dieses Verhalten zu erreichen, werden zufällige Ereignispunkte mit einem bestimmten Mindestabstand zueinander generiert. Die vorgegebene Gesamtanzahl von Iterationen wird in Intervalle mit der Länge des Mindestabstands aufgeteilt. Beispielweise kann bei einer Gesamtzahl von $N = 10$ und einem Mindestabstand von $d_{min} = 3$ die möglichen Ereignispunkte $[2, 5, 8]$ erzeugt werden (siehe Abbildung 8.10).

Die gegebenen Gesamtiterationen werden entsprechend den zuvor berechneten Ereignispunkten in Intervalle aufgeteilt. Bei einer Gesamtanzahl von $N = 10$ und den Ereignispunkten ergeben sich die Intervalllängen von $[3, 3, 3, 1]$. Diese Methode sorgt dafür, dass die Simulation realistische Muster und Verhaltensweisen widerspiegelt und somit eine Grundlage für die Bewertung und Anpassung von Systemen bietet.

Schritt 5: Zuweisung eines Zustandes zu jedem Intervall

Der abschließende Schritt umfasst die Zuordnung eines Zustandes zu jedem Intervall unter Berücksichtigung der angestrebten Wahrscheinlichkeitsverteilung.

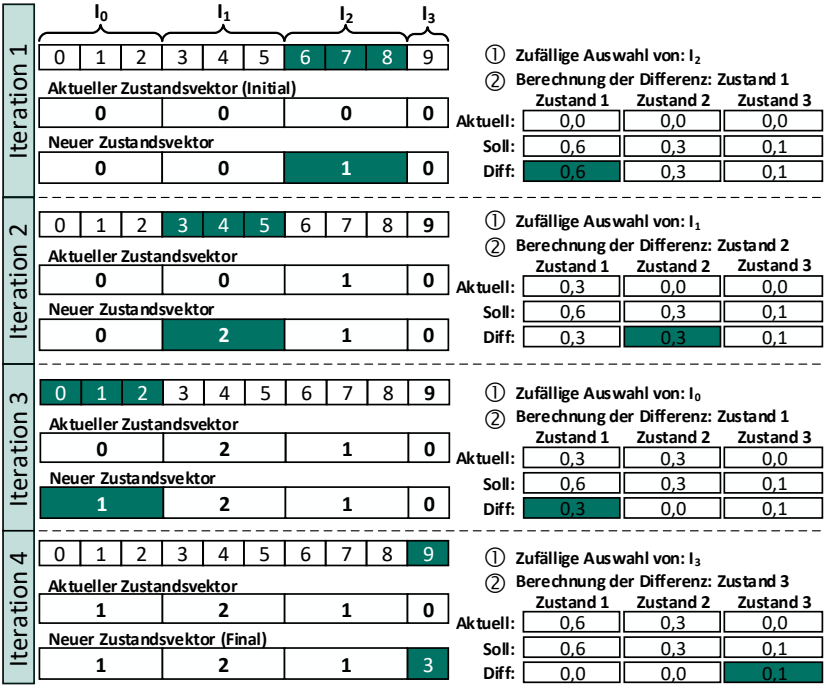

Abbildung 8.11: Darstellung des 4. Schrittes, der Zuweisung von Zuständen zu jedem Intervall

In Abbildung 8.11 im linken Bereich sind zehn Datenpunkte und vier Intervalle $[I_0]$ bis $[I_3]$ dargestellt. Diesen Intervallen werden iterativ die Zustände 1 bis 3 zugewiesen[1]. Die Anzahl der Durchläufe entspricht der Anzahl der Intervalle. In der ersten Iteration wird der aktuelle Zustandsvektor mit 0 in-

[1] Der Zustandsraum umfasst drei Zustände, da Zustand 4 in diesem Beispiel eine Wahrscheinlichkeit von 0% hat.

itialisiert. Zu Beginn wird ein zufälliges Intervall aus der Menge $[I_0, I_1, I_2, I_3]$ ausgewählt, in diesem Beispiel das Intervall I_2. Anschließend wird die Soll-Zustandsverteilung, wie in Schritt 3 beschrieben, mit der aktuellen Zustandsverteilung verglichen und die Differenz ermittelt. Der Zustand mit der größten Differenz, in diesem Beispiel Zustand 1, wird ausgewählt und dem Intervall I_2 zugewiesen.

In der zweiten Iteration wird der aktualisierte Zustandsvektor aus dem vorherigen Schritt übernommen. Erneut wird zufällig ein Intervall aus der Menge $[I_0, I_1, I_3]$ ausgewählt und der größte Unterschied zwischen der aktuellen und der Soll-Zustandsverteilung ermittelt. Bei zwei gleich identischen Differenzen erfolgt die Auswahl eines Zustandes zufällig (im Beispiel Zustand 2 in Abbildung Abbildung 8.11, Schritt 2). Dadurch wird der Wert von Zustand 2 dem Intervall I_1 zugeordnet.

Dieser Prozess wiederholt sich, bis jedem Intervall gemäß der Verteilung ein Zustand zugewiesen wurde. In Iteration 4 ist der finale Zustandsvektor dargestellt. Den Intervallen $[I_0, I_1, I_2, I_3]$ mit der Länge $[3, 3, 3, 1]$ wurden die Zustände $[1, 2, 1, 3]$ zugeordnet, wodurch eine Zustandsverteilung $[0.6, 0.3, 0.1]$ erreicht wurde. Das illustrierte Verfahren wurde am Beispiel der Sitzheizung durchgeführt, findet jedoch identisch auch für die in Tabelle Tabelle 8.4 aufgeführten Benutzeraktionen Anwendung. In Abbildung 8.12 sind die erzeugten Benutzeraktionen durch die kontextbasierte Benutzerlogik abgebildet.

Mithilfe der von CAGEN generierten Kontextdaten (siehe Abbildung 8.9) und Benutzeraktionen werden die Testdaten für die definierten Wegzwecke (siehe Abbildung 8.12) erstellt. Diese synthetisch erzeugten Testdaten dienen dazu, drei ML-Modelle (siehe Abschnitt 8.4) zu trainieren, die als selbstlernende Funktionen unter Einbezug von Benutzeraktionen fungieren.

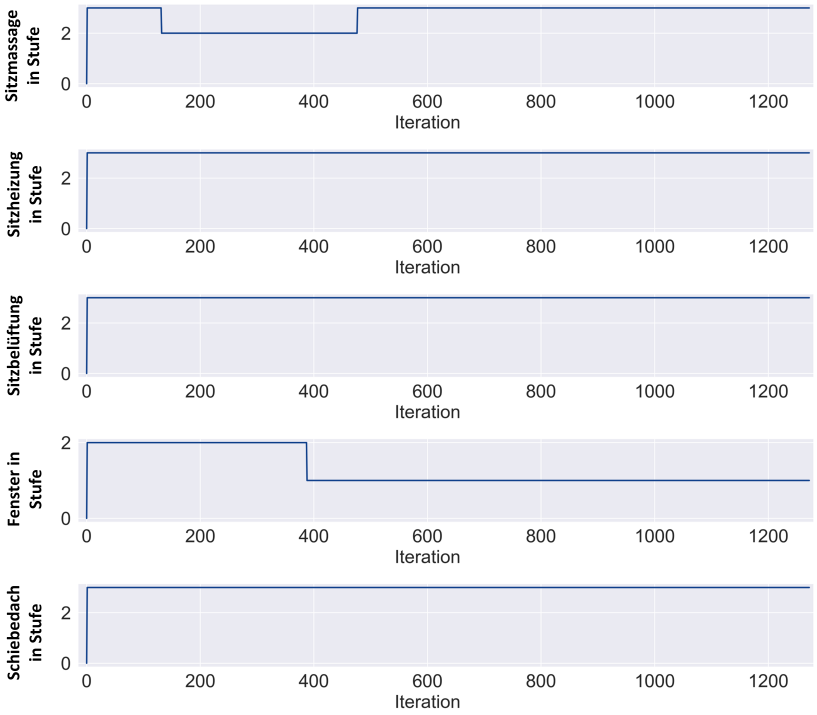

Abbildung 8.12: Darstellung der erzeugten Benutzeraktionen anhand der kontextbasierten Benutzerlogik für Wegzweck „Arbeit“ (siehe Abbildung 5.2, Bereich: Szenarioerzeugung, SE3)

8.4 ML-Modelle zum Erlernen von Benutzeraktionen

Zur Demonstration der Ganzheitlichkeit der Testmethode werden im Folgenden drei unterschiedliche ML-Modelle vorgestellt und in Kapitel 10 validiert. Die Betrachtung von mehreren SSIBs zeigt, dass die vorgeschlagene metamorphe Testmethode auf verschiedene Typen selbstlernender Systeme anwendbar ist.

Die ML-Modelle nutzen einen überwachten Lernansatz (siehe Abschnitt 2.1) und sind darauf ausgelegt, auf Basis der szenariobasierten Testdaten benutzerspezifische Ereignisse zu identifizieren (siehe Abbildung 8.13). Die ML-Modelle – das AC-Modell, KNN und KNN+ – weisen hinsichtlich der grundlegenden Nutzung der Trainingsdaten keine Unterschiede auf, da sie alle auf einem modellbasierten Ansatz beruhen (siehe Abbildung 2.2). Anstatt einzelne Instanzen direkt zu klassifizieren, erstellen sie allgemeine Modelle, die Muster in den Daten erkennen und auf dieser Grundlage Vorhersagen treffen.

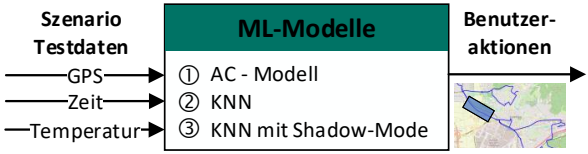

Abbildung 8.13: Selbstlernende Funktionen mit Benutzeraktionen (siehe Abbildung 5.2, Bereich: Evaluation, ML1)

Als SuT werden die drei folgende ML-Modelle betrachtet:

① Autonomous Comfort Modell (AC-Modell)

Das von Mercedes-Benz für die S-Klasse entwickelte AC-Modell ist ein maschinelles Lernmodell, das in dieser Dissertation zur Validierung zur Verfügung gestellt wird. Das Modell ist in das Mercedes-Benz User Experience System (MBUX) integriert und identifiziert die Routinen des Fahrers. Derzeit automatisiert das Modell das Sitzsystem, einschließlich Lüftung, Heizung und Massagefunktionen, mit Plänen zur schrittweisen Erweiterung auf weitere Innenraumsysteme. Das AC-Modell operiert als Black-Box-System (siehe Abschnitt 2.3), sodass interne Parameter nicht angepasst werden können. Die Eingabedaten für das Modell umfassen geografische Koordinaten (Latitude

und Longitude), Zeit- und Temperaturwerte. Das AC-Modell basiert auf einem Gaussian Mixture Model (GMM). Ein GMM ist ein probabilistisches Modell, das zur Darstellung der Verteilung von Datenpunkten verwendet wird. Es basiert auf der Annahme, dass die Datenpunkte aus einer Kombination mehrerer multivariater Gauß-Verteilungen generiert werden.

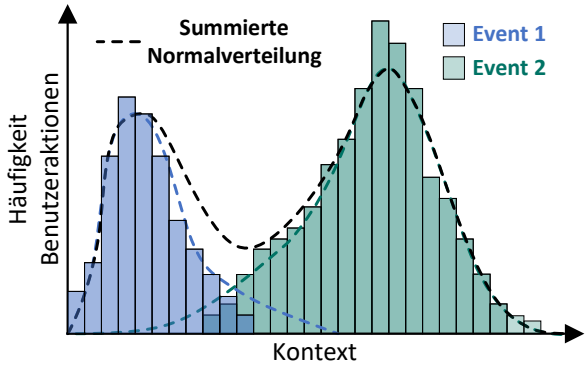

Abbildung 8.14: Darstellung der Histogramme von zwei Benutzeraktionen und die summierte Normalverteilung durch ein GMM.

Die Abbildung 8.14 zeigt zwei Histogramme eines GMMs im Kontext der Analyse von Benutzeraktionen. Auf der x-Achse ist der „Kontext“ dargestellt, während die y-Achse die „Häufigkeit der Benutzeraktionen“ anzeigt. Die Datenpunkte sind in Form von Histogrammen abgebildet, wobei zwei unterschiedliche Ereignisse (Event 1 und Event 2) durch verschiedenfarbige Balken (blau und grün) repräsentiert werden. Jede Gruppe von Balken entspricht einer Gruppe von Benutzeraktionen, die im gleichen Kontext auftreten. Die blau gefärbten Balken repräsentieren die Häufigkeit von Event 1, und die grün gefärbten Balken repräsentieren die Häufigkeit von Event 2.

Die beiden einzelnen Gauss’schen Verteilungen (blaue und grüne gestrichelte Linien) repräsentieren zwei verschiedene Komponenten des GMM. Diese Komponenten modellieren die Verteilungen der Benutzeraktionen für die beiden unterschiedlichen Ereignisse. Die summierte Normalverteilung (schwarze gestrichelte Linie) ist die gewichtete Summe der einzelnen Komponenten und stellt die gesamte Wahrscheinlichkeitsverteilung der Benutzeraktionen dar. Ein GMM verwendet diese summierte Verteilung, um die Wahrscheinlichkeit zu

berechnen, dass eine bestimmte Benutzeraktion im gegebenen Kontext auftritt. Benutzeraktionen im gleichen Kontext erhöhen die Wahrscheinlichkeit einer automatischen Aktivierung, indem sie das GMM befähigen, wiederkehrende Muster und Präferenzen präzise zu identifizieren und darauf basierend automatisch zu reagieren.

② KNN-Modell (KNN)

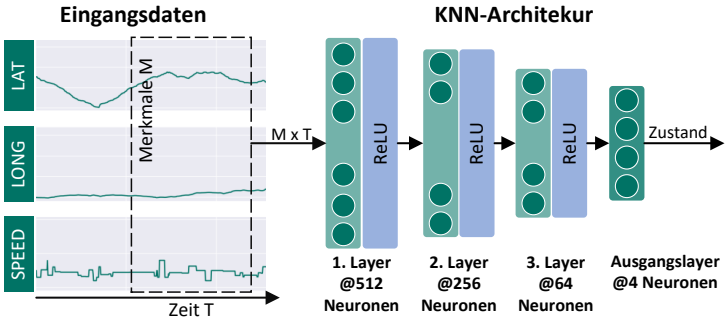

Abbildung 8.15: Darstellung der durch CAGEN erstellte Daten und die Architektur des KNNs.

Zusätzlich zu dem vorgestellten AC-Modell wird in dieser Dissertation ein auf KNNs basierendes ML-Modell (siehe Unterabschnitt 2.2.1) betrachtet. Der Einsatz mehrerer ML-Modelle dient dazu, die allgemeine Anwendbarkeit der Testmethode zu demonstrieren und ermöglicht einen direkten Vergleich verschiedener Ansätze. Das KNN-Modell wird als White-Box-Modell (siehe Abschnitt 2.3) implementiert und bietet im Gegensatz zum AC-Modell Zugriff auf Hyperparameter, wodurch eine gezielte Anpassung und Optimierung ermöglicht wird. Die Architektur, insbesondere die Anzahl der Neuronen, des KNN-Modells wurde mittels Grid-Search optimiert, wobei die Anzahl der Layer, Neuronen sowie die Aktivierungsfunktion variiert wurden. Dabei wurden Zweierpotenzen für die Neuronenzahl gewählt, da diese besonders effizient mit der Speicherarchitektur moderner Computer arbeiten.

Die Abbildung 8.15 zeigt die Architektur des KNN zur Verarbeitung der durch CAGEN erzeugten Kontextdaten. Diese Daten werden beispielhaft durch die drei Merkmale Latitude (LAT), Longitude (LONG) und Geschwindigkeit (SPEED) dargestellt. Die Daten dienen als Input des KNNs, dessen Architek-

tur auf der rechten Seite der Abbildung Abbildung 8.15 gezeigt ist. Zunächst werden die Merkmale M und die Zeit T als Matrix in das Netz eingeführt. Die erste Schicht des Netzwerks, der erste versteckte Layer, besteht aus 512 Neuronen. In diesem Layer werden die Eingangsdaten mithilfe der Rectified Linear Unit (ReLU)-Aktivierungsfunktion transformiert, um nicht-lineare Zusammenhänge zu modellieren. Anschließend werden die Daten an den zweiten versteckten Layer weitergeleitet, der aus 256 Neuronen besteht und ebenfalls die ReLU-Aktivierungsfunktion verwendet.

Dies ermöglicht eine weitere nicht-lineare Transformation der Eingangsdaten. Danach durchlaufen die Daten den dritten versteckten Layer, der aus 64 Neuronen besteht. Die Ausgangsschicht besteht aus vier Neuronen, die den Zustand in Form von Benutzeraktionen (siehe Abschnitt 8.3) ausgeben, der das Ergebnis der Verarbeitung der Eingangsdaten darstellt. Zur Optimierung des Modells wird das Gradientenabstiegsverfahren (siehe Unterunterabschnitt 2.2.2) verwendet, das durch die iterative Anpassung der Gewichte (siehe Definition 2.2)auf Basis des Gradienten der Kostenfunktion eine Minimierung des Fehlers (siehe Gleichung 2.6) ermöglicht.

③ KNN mit Shadowmode (KNN+)

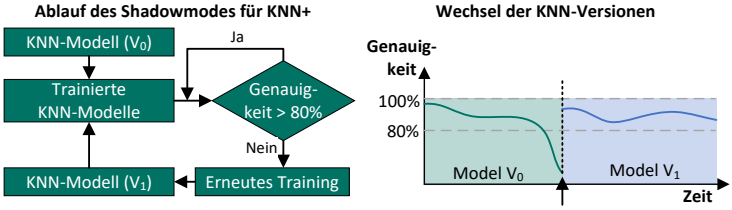

Abbildung 8.16: Entscheidungslogik des Shadow-Modus für KNN+ und Zusammenhang mit der beobachtbaren Genauigkeit des ML-Modells

Das dritte Modell in dieser Dissertation ist eine Erweiterung des KNNs, das um eine interne Logik im Shadow-Modus ergänzt wurde. Der Shadow-Modus ermöglicht es dem KNN, im Hintergrund zu operieren, ohne direkt in Prozesse einzugreifen oder Kontrollfunktionen auszuüben. Diese Methode wird vor allem in der Testphase neuer Modelle oder bei der Implementierung von Updates eingesetzt, um die Leistung des Modells zu bewerten, ohne die operationelle Integrität bestehender Systeme zu beeinträchtigen.

Die linke Seite der Abbildung 8.16 zeigt den Ablauf des Shadow-Modus für das erweiterte KNN (KNN+). Hier wird der Prozess visualisiert, bei dem das KNN-Modell V_0 mit neuen Daten trainiert wird. Das trainierte KNN-Modell wird dann hinsichtlich seiner Genauigkeit überprüft. Wenn die Genauigkeit des Modells über 80% liegt, wird es als akzeptabel angesehen und kann weiterverwendet werden. Falls die Genauigkeit jedoch unter 80% fällt, wird das Modell mit aktualisierten Trainingsdaten erneut trainiert. Dieser Zyklus wiederholt sich, bis ein Modell die geforderte Genauigkeit erreicht. Auf der rechten Seite der Abbildung 8.16 wird der Wechsel der KNN-Versionen im Laufe der Zeit dargestellt. Die y-Achse repräsentiert die Genauigkeit des Modells, während die x-Achse die Zeit darstellt. Zu Beginn ist das Modell V_0 aktiv. Wenn die Genauigkeit dieses Modells unter die Schwelle von 80% fällt, wird im Shadow-Modus ein neues Modell V_1 trainiert und ersetzt das ältere Modell V_0.

Die Implementierung des Shadow-Modus bietet mehrere Vorteile. Erstens ermöglicht sie eine kontinuierliche Überwachung und Bewertung der Modellgenauigkeit, ohne den laufenden Betrieb zu stören. Durch das Arbeiten im Hintergrund kann das KNN-Modell kontinuierlich aktualisiert und verbessert werden, wodurch die Genauigkeit und Zuverlässigkeit des Systems gesteigert werden.

Zweitens erlaubt der Shadow-Modus, dass sich Benutzeraktionen im Laufe der Zeit ändern können. Diese Veränderungen können die Genauigkeit des KNN-Modells beeinflussen, da die zugrunde liegenden Muster und Präferenzen der Benutzer nicht statisch sind. Der Shadow-Modus ermöglicht es, solche Veränderungen in Echtzeit zu erkennen und das Modell entsprechend anzupassen. Durch kontinuierliches Training und die Evaluierung neuer Modelle im Shadow-Modus ist das System in der Lage, auf veränderte Benutzerverhalten zu reagieren und eine Genauigkeit von über 80% aufrechtzuerhalten. Dieses Verhalten wird in Tabelle 10.2 (ebenso in Tabelle 12.1 und Tabelle 12.2) veranschaulicht, die für das KNN+ Modell Genauigkeitswerte von über 80% dokumentieren.

9 Metamorphes Testen und die Identifikation metamorpher Beziehungen

MT stellt eine Testmethode dar, die auf dem Konzept der MB basiert (siehe 4.4). Diese Methode hat in den vergangenen Jahren in der Software-Validierung sowie der Testentwicklung an Bedeutung gewonnen und bietet eine Möglichkeit, die Korrektheit von Software-Systemen zu überprüfen, insbesondere wenn spezifizierte Erwartungswerte fehlen (siehe Abschnitt 4.3). MB basieren auf der Idee, dass bestimmte Beziehungen zwischen Eingabe- und Ausgabe-Daten (siehe 4.6) bestehen, die unabhängig von der internen Logik des Programms oder Systems gelten. Diese Beziehungen ermöglichen es, Veränderungen oder Transformationen der Eingabe vorzunehmen und die erwartete Veränderung in der Ausgabe abzuleiten. Anstatt die Richtigkeit der Ausgabe zu überprüfen, werden die Beziehungen zwischen den Eingaben und den transformierten Ausgaben überprüft (siehe Abbildung 4.5).

MB können sowohl für die Verifikation als auch für die Validierung verwendet werden, wobei der Unterschied im Kontext und Ziel ihrer Anwendung liegt. Bei der Verifikation werden MB eingesetzt, um die Korrektheit des Systems zu überprüfen und sicherzustellen, dass es die spezifizierten Anforderungen erfüllt. Die Verifikation konzentriert sich darauf, ob das System richtig implementiert ist (siehe 2.10). Bei der Validierung (siehe 2.12) hingegen werden MB verwendet, um das Verhalten des Systems unter verschiedenen Bedingungen zu überprüfen und sicherzustellen, dass es den tatsächlichen Anforderungen und Bedürfnissen der Benutzer gerecht wird [116].

9.1 Abstraktionsebenenmodell für die Erstellung von metamorphen Beziehungen für Benutzerverhalten

Das Abstraktionsebenenmodell 9.1 bildet die methodische Grundlage, um MB zu identifizieren. Dies ist relevant in Situationen, in denen keine vordefinierten oder domänenspezifischen MB vorhanden sind.

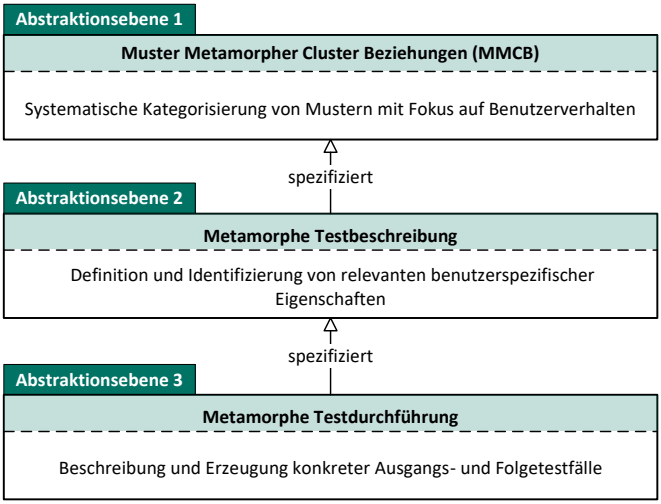

Abbildung 9.1: Entwickeltes Abstraktionsebenenmodell für Benutzerverhalten (siehe Abbildung 5.2, Bereich: Metamorphes Testen, MT2)

Die vorliegende Abbildung illustriert ein dreistufiges konzeptuelles Modell zur Entwicklung und Implementierung von metamorphen Testprozessen, die auf das Benutzerverhalten ausgerichtet sind. Auf der obersten Ebene, der Abstraktionsebene 1, wird das Konzept der „Muster Metamorpher Cluster Beziehungen“ eingeführt. Diese Ebene befasst sich mit der systematischen Kategorisierung (siehe Abbildung 9.2) solcher Muster, wobei ein besonderes Augenmerk auf das Verhalten von Benutzern gelegt wird.

Die Abstraktionsebene 2 vertieft das Konzept durch die „Metamorphe Testbeschreibung“. Es erfolgt eine Spezifizierung der auf Ebene 1 identifizierten

Muster durch die Definition und Identifizierung von relevanten, benutzerspezifischen Eigenschaften. Diese Ebene dient dazu, die Muster in testbare Hypothesen zu übersetzen, wobei die Charakteristika der Benutzer fokussiert werden. Ziel ist es, Eigenschaften (siehe Tabelle 9.1) zu entwickeln, die im Testprozess gemessen werden sollen, um die Auswirkungen der MB auf das Benutzerverhalten zu evaluieren.

Auf der untersten Abstraktionsstufe, der Ebene 3, erfolgt die „Metamorphe Testdurchführung“. In diesem Schritt werden die auf der zweiten Ebene definierten Eigenschaften und Hypothesen in konkrete Testfälle transformiert. Dabei stehen die Beschreibung und Generierung spezifischer Ausgangs- und Folgetestfälle im Mittelpunkt, welche die Basis für empirische Untersuchungen darstellen. Die vorhergehenden Abstraktionen werden hier operationalisiert und validiert, indem simulierte Szenarien konstruiert werden, in denen die Reaktionen der Benutzer auf Variationen beobachtet und analysiert werden können (siehe Abschnitt 10.2). Die Abstraktionsebene 3, die die eigentliche metamorphe Testdurchführung umfasst, wird gemeinsam mit der Evaluation in Kapitel 10 erörtert. Die Abbildung 9.1 stellt einen methodischen Ansatz dar, der von der theoretischen Konzeptualisierung bis hin zur praktischen Testimplementierung reicht. Die Pfeile zwischen den Ebenen symbolisieren den Prozess der Spezifizierung und Verfeinerung, der notwendig ist, um von allgemeinen Mustern zu individuellen Testverfahren zu gelangen, die aussagekräftige Einblicke in das Zusammenspiel von Benutzerverhalten und MB ermöglichen.

9.1.1 Abstraktionsebene 1: Muster Metamorpher Cluster Beziehungen

Die MMCB sind in dieser Dissertation definierte Relationstypen, die sich aus der systematischen Analyse von Clustern in Ausgangs- und Folgedatensätzen ergeben und auf dem Konzept der „abstrakten MB“ [117] basieren. MMCBs beziehen sich auf spezifische Testmuster innerhalb der metamorphen Testmethodik, die zur Validierung eines SSIB verwendet werden. Der Fokus von MMCBs liegt auf den Beziehungen zwischen verschiedenen Clustern von Testfällen, die durch MB definiert werden.

Vor der Durchführung der Clusteranalyse wird eine zweidimensionale Hauptkomponentenanalyse (PCA) auf die Testdaten angewendet. Diese methodische

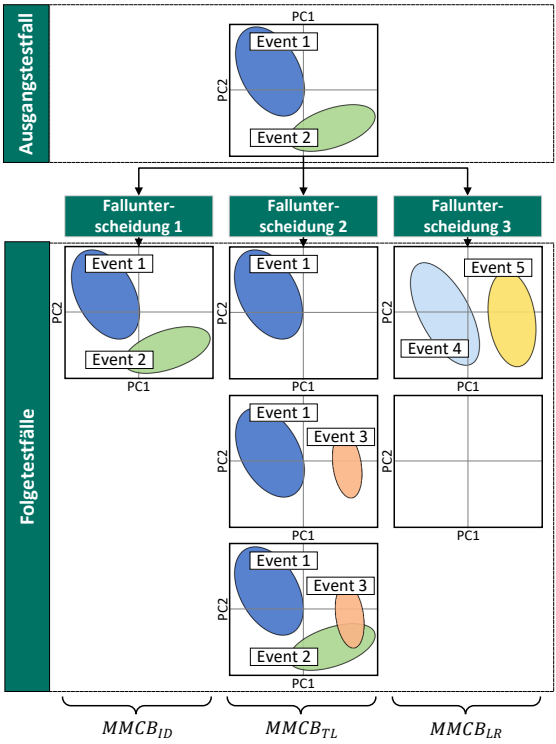

Abbildung 9.2: Unterteilung der MMCBs anhand einer Fallunterscheidung basierend auf der Clustervisualisierung. (siehe Abbildung 5.2, Bereich: Metamorphes Testen, AE1)

Vorstufe ist erforderlich, da die Testdaten der untersuchten Szenarien mehrdimensionale Datensätze umfassen, die sich nicht direkt visualisieren lassen. Durch die Implementierung der PCA (siehe Abbildung 10.1) wird eine Dimensionsreduktion erreicht, die es ermöglicht, Cluster basierend auf ähnlichen Benutzeraktionen zu identifizieren und zu visualisieren. Jedes Cluster umfasst das Merkmal einer Benutzeraktion, die einem spezifischen Ereignis zugeordnet ist, und wird demnach als Ereigniscluster spezifiziert.

Die MMCBs klassifizieren das Benutzerverhalten in drei disjunkte Kategorien: solche, die eine Identität, eine partielle Übereinstimmung und eine Nichtüber-

einstimmung zwischen den Ausgangs- und Folgetestfällen aufweisen (siehe 9.2). Diese Kategorien werden wie folgt definiert und kategorisiert:

Das Muster der Metamorphen Cluster Beziehungen $\mathrm{MMCB_{ID}}$ postuliert die Identität der Clusterpopulationen im Ausgangstestfall A_{out} im Vergleich zu jenen im Folgetestfall F_{out}.

Fallunterscheidung 1 *Für jedes Ereignискluster im Ausgangstestfall A_{out} existiert ein korrespondierendes Ereignискluster im Folgetestfall, das folgendes Identitätskriterium erfüllt:*

$$|m_s - m_f| < \varepsilon \tag{9.1}$$

Hierbei symbolisiert m_s den Mittelwert der Merkmale innerhalb des Ereignisclusters im Ausgangstestfall, m_f den Mittelwert der Merkmale innerhalb des korrespondierenden Ereignisclusters im Folgetestfall F_{out}, und ε eine vordefinierte Toleranzschwelle, die geringfügige Abweichungen zulässt.

Die zweite $\mathrm{MMCB_{TL}}$ impliziert, dass ein Teil der im Ausgangstestfall A_{out} beobachteten Ereignискluster im Folgetestfall F_{out} in ähnlicher Form wiedergefunden werden kann, was eine partielle Konsistenz des beobachtbaren Verhaltens zwischen den Datensätzen anzeigt. Somit gilt:

Fallunterscheidung 2 *Für mindestens ein Ereignискluster im Ausgangstestfall A_{out} existiert ein Ereignискluster im Folgetestfall F_{out}, das das Identitätskriterium 9.1 erfüllt.*

Die Metamorphe Cluster Beziehung $\mathrm{MMCB_{LR}}$ kennzeichnet eine Situation, in der zwischen den Clustern des Ausgangs- und des Folgetestfalls keine Identität festgestellt werden kann, was auf vollständig unterschiedliche oder nicht korrespondierende Clusterpaare hinweist.

Fallunterscheidung 3 *Für kein Ereignискluster im Ausgangstestfall A_{out} existiert ein Ereignискluster im Folgetestfall F_{out}, sodass für kein Clusterpaar das Identitätskriterium 9.1 bestimmt werden kann.*

Durch die Fallunterscheidung und die Auswertung der Ähnlichkeit mittels des Identitätskriteriums wird die Anforderung AM3 erfüllt, die eine automati-

sierte Analyse ermöglichen soll. Die Verwendung der Ereigniscluster wird in Abbildung 10.1 aufgegriffen und vertiefend anhand von Kontext und Benutzeraktionen und der in Abbildung 8.13 vorgestellten ML-Modelle erläutert.

9.1.2 Abstraktionsebene 2: Metamorphe Testbeschreibung

Auf der zweiten Ebene der Anwendung metamorpher Tests steht die Definition von Eigenschaften, welche das SuT aufweisen muss, im Mittelpunkt. Die Auswahl und Formulierung von aussagekräftigen MB ist für die Effektivität der Testdurchführung unerlässlich [108]. Diese ermöglichen es, Beziehungen und repräsentative Systemeigenschaften zu identifizieren, wodurch nicht nur die Funktionalität des Systems verifiziert, sondern auch potenzielle Fehlerquellen und Schwachstellen aufgedeckt werden können. Die Erarbeitung dieser Testfälle basiert auf den in der ersten Abstraktionsebene festgelegten MMCBs, wodurch eine methodische und effiziente Testausrichtung gewährleistet wird. Die Anzahl der MB stellt einen Faktor im Kontext des metamorphen Testens dar, da sie direkte Auswirkungen auf die Effektivität und Effizienz des Testprozesses hat. Gemäß den Erkenntnissen von Segura et al. [110], basierend auf einer Untersuchung von 119 wissenschaftlichen Veröffentlichungen, verwenden etwa 75% dieser Studien zwischen 1 und 14 MB (siehe Abbildung 9.3). In dieser Dissertation wird die Anzahl der verwendeten MB gemäß der Studie von Segura et al. [110] festgelegt und auf zehn begrenzt.

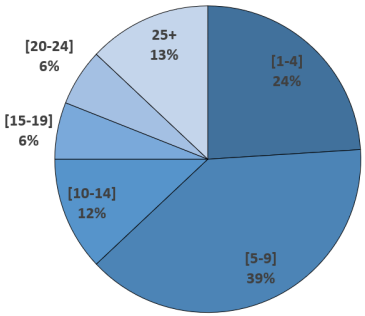

Abbildung 9.3: Prozentsatz der Anzahl der verwendeten MB basierend auf Segura et al. [110] auf Grundlage von 119 untersuchten wissenschaftlichen Veröffentlichungen.

Identifizierte Eigenschaften und Teilaspekte Im folgenden Kapitel liegt der Fokus auf der Definition und Identifizierung relevanter Eigenschaften selbstlernender Systeme, insbesondere im Zusammenhang mit Benutzeraktionen. Diese Eigenschaften bilden die Grundlage für die Definition von geeigneten MB, präzisiert durch MMCBs (siehe Unterabschnitt 9.1.1). Durch eine Bestimmung dieser Eigenschaften wird eine Analyse des SuT ermöglicht, um seine Effektivität und Anpassungsfähigkeit an den Kontext und die Benutzeraktion zu validieren.

Transition des Benutzerverhaltens

Die Eigenschaft der „Transition des Benutzerverhaltens“ umfasst die Überprüfung, ob das SuT in der Lage ist, Veränderungen im Verhalten des Benutzers zu erkennen. Diese Veränderungen resultieren nicht aus einer direkten Veränderung des Kontextes oder der Umwelt, sondern entstehen intrinsisch durch den Benutzer selbst. Ein konkretes Beispiel hierfür wäre die Situation, in der ein Benutzer, der zuvor geraucht hat, zum Nichtraucher wird und dadurch sein Verhalten in Bezug auf das Öffnen und Schließen des Fensters ändert. Die Eigenschaft der „Transition des Benutzerverhaltens“ lässt sich in die folgenden Teilaspekte unterteilen: das Hinzufügen neuer Verhaltensmuster und das Entfernen bekannter Verhaltensmuster. Das Hinzufügen neuer Verhaltensmuster bezieht sich auf die Fähigkeit des SuT, ein neues Verhalten des Benutzers zu erkennen und entsprechend darauf zu reagieren. Dies ermöglicht dem SuT, sich an die Veränderungen im Benutzerverhalten anzupassen und angemessene Interaktionen oder Dienste anzubieten ($MMCB_{TL}1.1$). Das Entfernen bekannter Verhaltensmuster bezieht sich auf die Fähigkeit des SuT, das Wegfallen eines bisherigen Verhaltensmusters des Benutzers zu identifizieren und entsprechende Anpassungen vorzunehmen ($MMCB_{ID}1.2$). Durch die Berücksichtigung dieser Teilaspekte wird die Fähigkeit des SuT zur adäquaten Erfassung und Handhabung von Veränderungen im Benutzerverhalten gewährleistet.

Zusätzlich zu den Teilaspekten des Erlernens und Verlernens von Benutzerverhalten soll das SSIB den Teilaspekt der Differenzierungsfähigkeit besitzen ($MMCB_{TL}1.3$). Die Differenzierungsfähigkeit ist von Bedeutung, da sie es dem SuT ermöglicht, personalisierte Interaktionen und Dienste anzubieten. Jeder Benutzer des selbstlernenden Systems kann Vorlieben, Gewohnheiten und Bedürfnisse besitzen, die vom SuT er-

kannt und automatisch berücksichtigt werden sollen. Durch die Durchführung dieses Testansatzes lässt sich überprüfen, ob das selbstlernende System die Fähigkeit besitzt, zwischen Nutzern mit verschiedenen Verhaltensweisen zu differenzieren und für jeden Anwender individuelle Routinen zu extrahieren.

Transition der Umgebungsbedingungen
Eine Veränderung des Benutzerverhaltens kann im Gegensatz zum vorherigen Aufzählungspunkt, der „Transition des Benutzerverhaltens“, ebenfalls durch Veränderungen in der Umwelt induziert werden. Die Umwelt hat einen unmittelbaren Einfluss auf das Verhalten des Benutzers und führt dadurch indirekt zu Anpassungen in den Routinen. Die Eigenschaft „Transition der Umgebungsbedingungen“ kann durch die folgenden drei Teilaspekte definiert werden: die Modifikation der Zeitmerkmale, das Erkennen von saisonalen Effekten sowie das Erkennen der Fahrtrichtung. Durch den erstgenannten Teilaspekt „die Modifikation der Zeitmerkmale“, ist das selbstlernende System in der Lage, die Veränderungen in der Zeit wahrzunehmen und entsprechend zu reagieren. Ein beispielhafter Fall könnte darin bestehen, dass das selbstlernende System Änderungen in der Uhrzeit erkennt und daraufhin ihre Aktivitäten oder Einstellungen entsprechend anpasst ($MMCB_{LR}2.1$). Der zweite Teilaspekt ist das Erkennen von saisonalen Effekten. Das selbstlernende System ist in der Lage, saisonale Veränderungen zu erkennen und darauf zu reagieren. Dies kann beispielsweise bedeuten, dass das selbstlernende System das Verhalten in Bezug auf Heizung oder Klimaanlage an die jeweilige Jahreszeit anpasst ($MMCB_{TL}2.2$). Der dritte Teilaspekt ist das Erkennen der Fahrtrichtung. Das selbstlernende System ist in der Lage, die Fahrtrichtung des Benutzers zu erkennen und darauf zu reagieren. Das heißt, dass das selbstlernende System feststellen kann, ob sich der Benutzer von zu Hause zur Arbeit bewegt oder umgekehrt. Basierend auf dieser Information kann das System entsprechende Empfehlungen, Einstellungen oder Interaktionen anbieten, die den Bedürfnissen des Benutzers während der Fahrt entsprechen ($MMCB_{TL}2.3$).

Robustheit und Ausreißererkennung Die Eigenschaft der Robustheit bezieht sich auf die Fähigkeit eines Algorithmus, mit Ungenauigkeiten und Störungen umzugehen, die durch die Sensoren bei der Erfassung und Darstellung der Umgebung entstehen. Die erfassten Sensordaten

sind durch Störsignale (Rauschen) überlagert, die die Präzision der gemessenen Daten beeinflussen können. Ein robuster Algorithmus sollte in der Lage sein, Störungen zu erkennen und zu reduzieren, um dadurch zuverlässige Ergebnisse zu liefern ($MMCB_{ID}3.1$). Zusätzlich zur Robustheit stellt die Ausreißererkennung ein Teilaspekt bei der Verarbeitung von Daten dar. Ausreißer sind Datenpunkte, die von der erwarteten Datenverteilung abweichen und potenziell fehlerhaft sind. Eine SSIB soll in der Lage sein, sowohl das Rauschen als auch Ausreißer zu erkennen und angemessen zu behandeln, um die Qualität der Daten und die Genauigkeit der Modelle zu optimieren ($MMCB_{ID}3.2$).

Durch die Definition der Eigenschaften und Teilaspekte kann eine Validierung dieser erfolgen, wodurch die Anforderung AM2 erfüllt wird. Die Teilaspekte, symbolisiert durch die IDs der MMCBs, werden entsprechend der dritten Abstraktionsebene (siehe Abbildung 9.1) in konkrete Ausgangs- und Folgetestfälle (siehe Abschnitt 10.2) umgesetzt.

Eigenschaften	Teilaspekte	ID
Transition des Benutzerverhaltens	Erlernen von Benutzerverhalten	$MMCB_{TL}1.1$
	Verlernen von Benutzerverhalten	$MMCB_{ID}1.2$
	Differenzieren zwischen Verhalten von Benutzern	$MMCB_{TL}1.3$
Transition des Umgebungsverhaltens	Modifikation des Zeitmerkmals	$MMCB_{LR}2.1$
	Erkennen von saisonalen Effekten	$MMCB_{TL}2.2$
	Erkennen der Fahrtrichtung	$MMCB_{TL}2.3$
Robustheit und Ausreißererkennung	Funktionsfähigkeit mit Messrauschen	$MMCB_{ID}3.1$
	Ignorieren von Ausreißern	$MMCB_{ID}3.2$

Tabelle 9.1: Übersicht der identifizierten Eigenschaften und Teilaspekte des SSIB (siehe Abbildung 5.2, Bereich: Metamorphes Testen, AE2)

Nach der systematischen Kategorisierung der Muster von Benutzeraktionen auf Abstraktionsebene 1 sowie der Definition und Identifikation benutzerspezifischer Eigenschaften auf Abstraktionsebene 2, wird die metamorphe Testdurchführung in folgendem Kapitel 10 separat behandelt.

10 Abstraktionsebene 3: Metamorphe Testdurchführung und Evaluation

Das Abstraktionsebenenmodell (siehe Abschnitt 9.1) für MMCBs beschreibt in der ersten Ebene „Muster Metamorpher Cluster Beziehungen“ (siehe Abbildung 9.2) typische Muster, die bei der Validierung im Rahmen des Metamorphen Testens auftreten können. Die zweite Ebene „Metamorphe Testbeschreibung“ (siehe Tabelle 9.1) legt die Eigenschaften fest, die das SuT erfüllen muss. Die in diesem Kapitel vorgestellte dritte Ebene, „Metamorphe Testdurchführung“, nutzt konkrete Eingangs- und Folgetestfälle, um die Funktionalität anhand der in Ebene 2 definierten Eigenschaften zu prüfen, indem typische Muster, wie in Ebene 1 beschrieben, zur Validierung des Systems erkannt werden.

10.1 Visualisierung der Testergebnisse durch Cluster

Die Eingangstestfälle (siehe Abbildung 10.1) umfassen sowohl Kontextdaten (Lat/Long, Speed, Zeit) als auch Benutzeraktionen (Fenster, Sitzheizung, Sitzmassage). Diese Daten werden in eine PCA-Darstellung überführt, die die Datenpunkte in einem zweidimensionalen Raum anhand zwei künstlicher Achsen (PC1 und PC2) visualisiert:

- PC1 (Hauptkomponente 1): PC1 ist die Richtung, in der die Varianz der Daten am größten ist. Das bedeutet, dass PC1 die Achse ist, entlang derer die Daten die meiste Streuung aufweisen. Dadurch erfasst PC1 Muster und die größte Veränderung in den Daten.
- PC2 (Hauptkomponente 2): PC2 ist die zweite Achse, die die zweitgrößte Varianz der Daten beschreibt, jedoch orthogonal (senkrecht) zu PC1 ist.

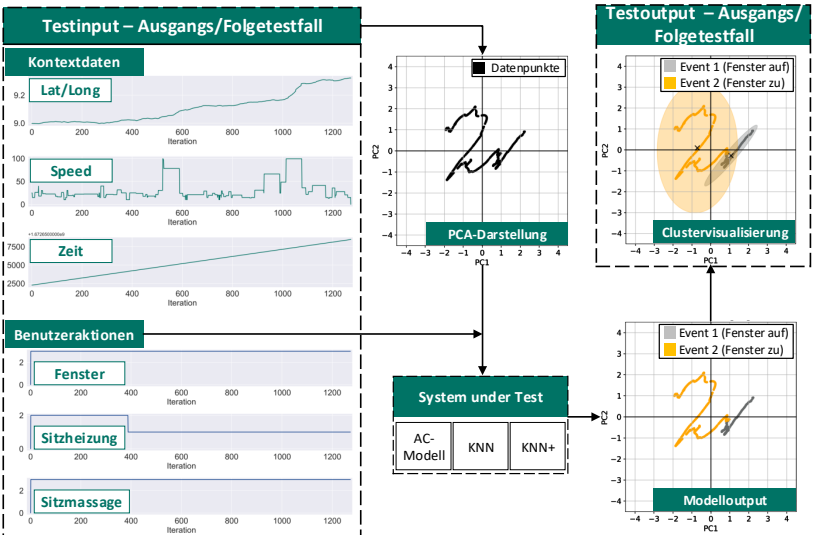

Abbildung 10.1: Erklärung der Clustervisualisierung für einen Testinput (z.B. Eingangstestfall bzw. Folgetestfall) (siehe Abbildung 5.2, Bereich: Evaluation, EV1)

> Das bedeutet, PC2 steht in keinem Zusammenhang zu PC1 und erfasst zusätzliche Variationen, die nicht bereits durch PC1 beschrieben werden.

Im nächsten Schritt werden die Daten an die SSIBs übergeben, das aus drei selbstlernenden Modellen besteht: dem AC-Modell, KNN und der erweiterten Version KNN+ (siehe Abschnitt 8.4). Diese ML-Modelle klassifizieren die Datenpunkte auf Basis der erkannten Benutzeraktionen. Der Modelloutput visualisiert die klassifizierten Datenpunkte und hebt die verschiedenen Benutzeraktionen farblich hervor, beispielsweise in Bezug auf das Öffnen und Schließen von Fenstern.

Der finale Schritt des Prozesses ist die Clustervisualisierung, die auf dem Modelloutput aufbaut. Anhand der klassifizierten Datenpunkte werden Cluster gebildet, die es ermöglichen, mehrere Testoutputs vergleichend zu analysieren. Diese Methode bietet den Vorteil, dass die Datenpunkte nicht isoliert betrachtet werden müssen, sondern in Gruppen analysiert werden können, was die

Aussagekraft der Ergebnisse erhöht und den Vergleich der unterschiedlichen Modelle erleichtert.

10.2 Testfälle für die Validierung mittels metamorpher Beziehungen

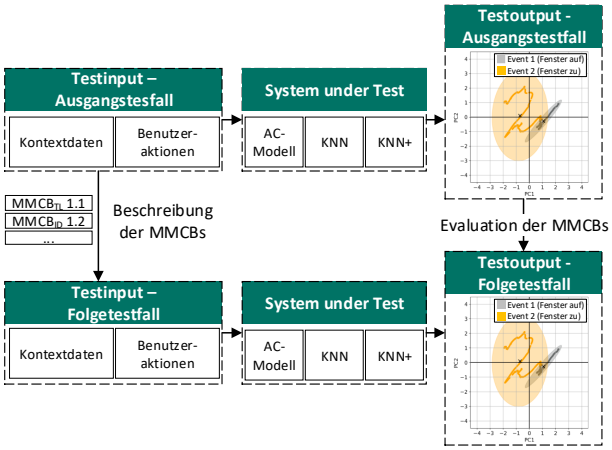

Abbildung 10.2: Durchführung und Validierung des Metamorphen Testens anhand der Clustervisualisierung nach dem Vorgehen aus Abbildung 10.1 (siehe Abbildung 5.2, Bereich: Evaluation, AE3)

Durch die Clustervisualisierung (siehe Abbildung 10.1) kann das Verhalten der SSIB für den jeweiligen Testinput dargestellt werden.

Die Metamorphe Testdurchführung entsprechend der Abstraktionsebene 3 evaluiert mehrere dieser Clustervisualisierungen miteinander (siehe Abbildung 10.2). Diese Visualisierungen werden in Relation zueinander gesetzt, um die MMCBs zu überprüfen und zu validieren. Für jede MMCB wird sowohl für den Ausgangstestfall als auch den Folgetestfällen eine Clustervisualisierung erstellt (siehe Abbildung 10.1).

Der Testinput basiert auf den definierten Wegzwecken „Arbeit“, „Erledigung“ und „Begleitung“ (siehe Unterabschnitt 6.2.1). Die in diesem Kapitel vorgestellte Testdurchführung und Evaluation konzentriert sich auf den Wegzweck

„Arbeit“ (siehe Abbildung 6.4). Die Evaluationsergebnisse zu den Wegzwecken „Erledigung“ und „Begleitung“ sind im Anhang auf den Seiten 202 und 211 aufgeführt.

10.2.1 $MMCB_{TL}1.1$ - Erlernen von Benutzerverhalten

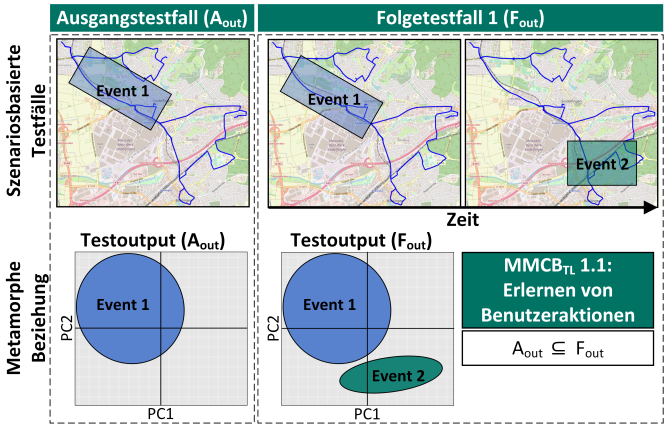

Abbildung 10.3: Beschreibung von Ausgangs- und Folgetestfall für $MMCB_{TL}1.1$ für das Erlernen von Benutzerverhalten

Dieser Testfall überprüft die Fähigkeit des SSIB, unterschiedliche Benutzerverhaltensweisen zu erlernen. Im Ausgangstestfall wird ein spezifisches Verhalten simuliert, dargestellt durch Event 1 in Abbildung 10.3. Der Folgetestfall erweitert diese Simulation um ein zusätzliches Verhalten, repräsentiert durch Event 2 in Abbildung 10.3.

Das zwischen dem Ausgangs- und dem Folgetestfall erwartete Verhalten ist durch $MMCB_{TL}1.1$ definiert: die Integration eines zusätzlichen Verhaltens (Event 2 in Abbildung 10.3) soll die Erzeugung eines entsprechenden Eventclusters zur Folge haben. Der Cluster für Event 1 soll dabei unverändert bleiben, da die Eventkategorien Stufe 1 und Stufe 2 eindeutig voneinander getrennt sind. Das Event 2 muss hingegen im Output F_{out} beobachtbar sein, damit die $MMCB_{TL}1.1$, die vom Muster Teilmenge ist, erfüllt ist.

Evaluation der $MMCB_{TL}$ 1.1

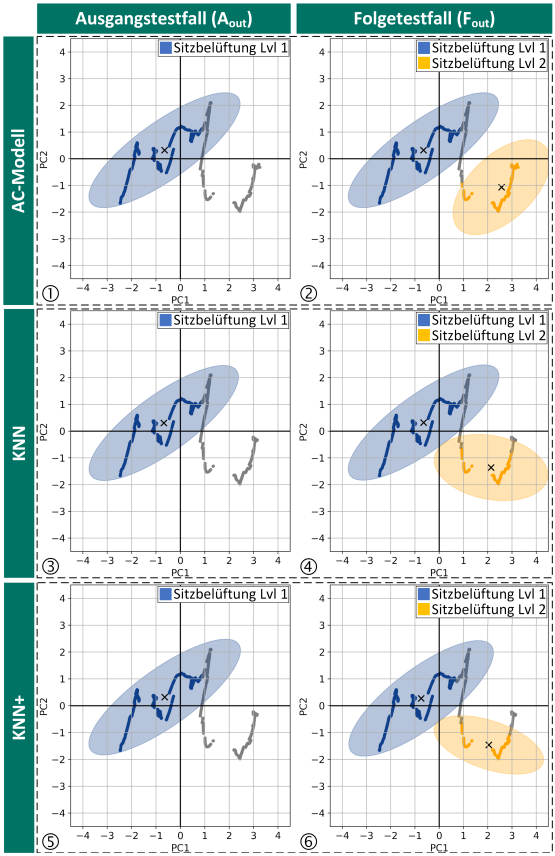

Abbildung 10.4: Evaluation der $MMCB_{TL}$ 1.1 für das Erlernen von Benutzerverhalten

Der Ausgangstestfall basiert auf dem Szenario „Arbeit“ und umfasst eine Benutzeraktion für die Sitzbelüftung der Stufe 1. Dieser Ausgangstestfall simuliert zehn aufeinanderfolgende Tage und beinhaltet durch CAGEN erzeugten Kontext und Benutzeraktionen. Die Anzahl der simulierten Tage hängt von der MMCB und der zu testenden Eigenschaft ab. Für die Validierung saisonaler Ef-

fekte (siehe $MMCB_{TL}2.2$) ist ein längerer Beobachtungszeitraum erforderlich, während bei MMCBs, die täglich wiederkehrende Routinen erlernen sollen, ein kürzerer Zeitraum ausreicht.

Der Folgetestfall modifiziert diesen Ausgangstestfall, indem ein zweites Event (Sitzbelüftung Stufe 2) hinzugefügt und ebenfalls über zehn Tage simuliert wird. Die Entscheidung über die Erfüllung der MB erfolgt durch den Vergleich der Testoutputs des Ausgangstestfalls und des Folgetestfalls. Für diesen Testfall bedeutet dies, dass die $MMCB_{TL}1.1$ erfüllt ist, wenn der Eventcluster für Sitzbelüftung der Stufe 1 zwischen Ausgangs- und Folgetestfall unverändert bleibt und gleichzeitig im Folgetestfall ein Eventcluster für Sitzbelüftung Stufe 2 erkennbar ist.

Die Analyse der Testoutputs (siehe Abbildung 10.4) aller drei untersuchten ML-Modelle zeigt das gewünschte Verhalten: sie zeigen einen Eventcluster für die Sitzbelüftung Stufe 2 (nur in F_{out}) und behalten den Eventcluster der Sitzbelüftung Stufe 1 (sowohl in A_{out} als auch in F_{out}) bei. Somit erfüllen alle drei untersuchte ML-Modelle die Anforderungen der $MMCB_{TL}1.1$.

Die Ähnlichkeit der Eventcluster der Sitzbelüftung Stufe 1 in A_{out} und F_{out} kann nicht nur visuell bestimmt werden, sondern auch durch die Berechnung des euklidischen Abstandes der Ellipsenzentren (siehe Tabelle 12.3). Der euklidische Abstand der Ellipsenzentren beträgt für das AC-Modell (Abbildung 10.4, Unterabbildung 1 und 2) 0.031, für das KNN 0.16 und für das KNN+ 0.12 (siehe Tabelle 12.3). Diese Werte bestätigen die visuelle Betrachtung, dass der Eventcluster der Stufe 1 der Sitzbelüftung keine Veränderung zwischen A_{out} und F_{out} erfährt.

10.2.2 $MMCB_{ID}1.2$ - Verlernen von Benutzerverhalten

Die $MMCB_{ID}1.2$ basiert auf der vorherigen $MMCB_{TL}1.1$ und untersucht, die Fähigkeit des SuT bereits erlernte Verhaltensmuster zu vergessen. Diese Fähigkeit ist von grundlegender Bedeutung, da sich Benutzerverhaltensweisen im Laufe der Zeit ändern können (siehe Abbildung 10.5). Die untersuchten ML-Modelle müssen daher nicht nur neue Verhaltensmuster erlernen (wie in $MMCB_{TL}1.1$), sondern auch in der Lage sein, veraltete Muster zu vergessen. Dies stellt sicher, dass die SuT aktualisierbar sind und sich an die sich weiterentwickelnden Präferenzen und Bedürfnisse der Benutzer anpassen können.

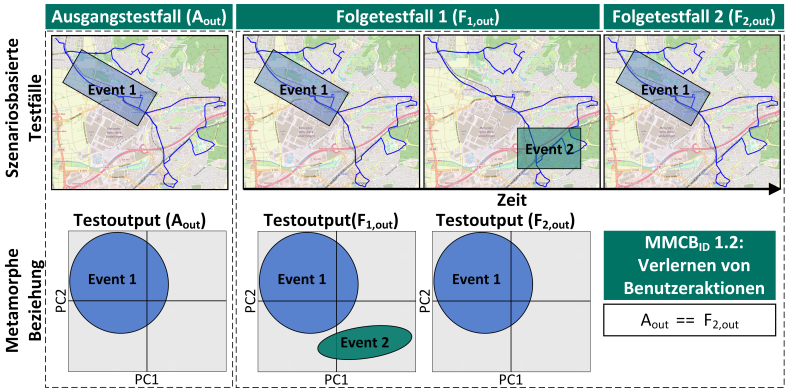

Abbildung 10.5: Beschreibung von Ausgangs- und Folgetestfälle für MMCB$_{ID}$1.2 für das Verlernen von Benutzerverhalten

Der Ausgangstestfall A_{out} bleibt unverändert im Vergleich zur MMCB$_{TL}$1.1, während der Folgetestfall die bereits etablierten Szenarien nutzt und ein zusätzliches Szenario einfügt, in dem das zuvor erlernte Verhalten nicht mehr auftritt. Hierbei wird ein Event 2 implementiert, um ein Verhaltensmuster zu simulieren, das das zu testende System verlernen soll. Die MMCB$_{ID}$1.2 fordert, dass die Testoutputs des Ausgangstestfalls A_{out} und des Folgetestfalls $F_{2,out}$ identisch sind. Sollte kein identisches Verhalten in der Clustervisualisierung erkennbar sein, ist das SuT nicht in der Lage, bereits erlernte Verhaltensweisen zu vergessen.

Evaluation der MMCB$_{ID}$1.2

Die MMCB$_{ID}$1.2 wird anhand der Massagefunktion evaluiert (siehe Abbildung 10.6). Der Ausgangstestfall A_{out} erlernt die Massagefunktion der Stufe 2 und der Stufe 3 über einen Zeitraum von 30 Tagen, wobei die die ersten 20 Tage identische zu MMCB$_{TL}$1.1 sind und mit weiteren 10 Tagen erweitert wurden. Im ersten Folgetestfall $F_{1,out}$ soll die Massagefunktion der Stufe 1 erlernt werden (siehe Abbildung 10.6, Unterabbildung 2,5 und 8). Der Testaufbau (A_{out} und $F_{1,out}$) entspricht dem der MMCB$_{TL}$1.1. Der zweite Folgetestfall $F_{2,out}$ erweitert diesen Testaufbau, indem das Massageevent der Stufe 1 wieder entfernt wird. Die Testdaten im Folgetestfall 2 $F_{2,out}$ umfassen ebenfalls 30

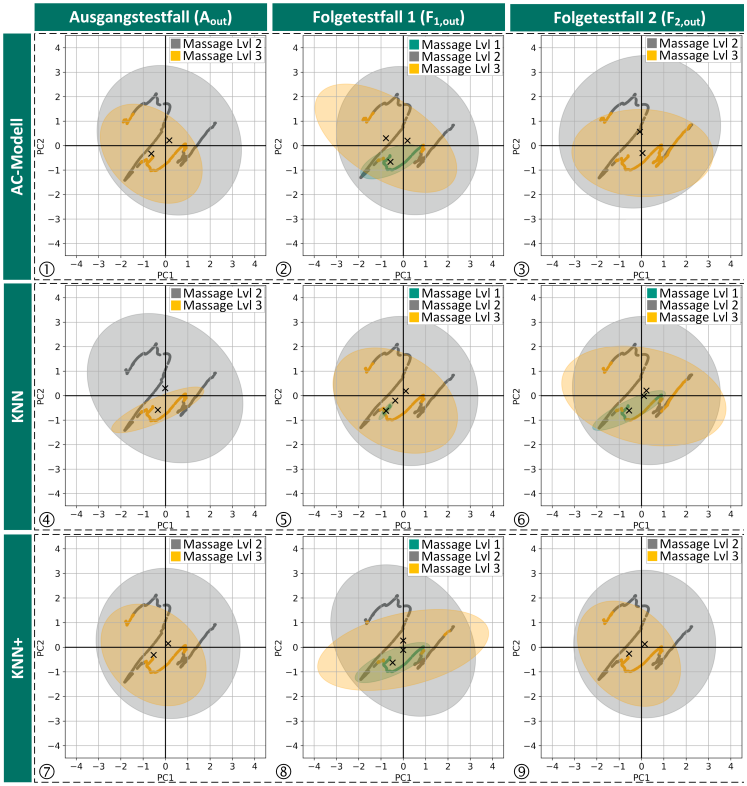

Abbildung 10.6: Evaluation der $MMCB_{ID}1.2$ für das Verlernen von Benutzerverhalten

Tage, wobei in den ersten 10 Tagen die Massagestufe Stufe 2,3, in den nächsten 10 Tagen die Massagestufe der Stufe 1,2 und 3 und in den letzten 10 Tagen nur die Massageeventstufe 2,3 ausgeführt werden. Damit das ML-Modell die $MMCB_{ID}1.2$ erfüllt, müssen der Ausgangstestfall A_{out} und der Folgetestfall 2 $F_{2,out}$ eine identische Clustervisualisierung aufweisen.

Bei der Evaluation (siehe Abbildung 10.6) der drei ML-Modelle ist beobachtbar, dass das KNN+ Modell die $MMCB_{ID}1.2$ erfüllt. Der Testoutput des Ausgangstestfalls A_{out} des KNN+ Modells (siehe Abbildung 10.6, Unterabbildung 7) ist identisch mit dem des Folgetestfalls 2 $F_{2,out}$ (siehe Abbildung 10.6,

Unterabbildung 9), womit die $MMCB_{ID}1.2$ vom Typ der Identität erfüllt ist. Die euklidische Distanz zwischen dem Eventcluster Massage Stufe 2 und 3 in A_out (Unterabbildung 7) und dem korrespondierenden Eventcluster Massage Stufe 2 und 3 in $F_{2,out}$ (Unterabbildung 9) beträgt 0.126 bzw. 0.179 (siehe Tabelle 12.3) und bestätigen damit die Identität.

Das AC-Modell und das KNN erfüllen die $MMCB_{ID}1.2$ nicht. Der Vergleich der Eventcluster von Unterabbildung 1 mit Unterabbildung 3 für das AC-Modell sowie der Vergleich der Unterabbildung 4 mit Unterabbildung 6 für das KNN-Modell zeigt keine Identität gemäß der Beschreibung (siehe Abbildung 10.5) für $MMCB_{ID}1.2$ und erfüllt nicht das Identitätskriterium (siehe Gleichung 9.1). Der Testoutput des AC-Modells zeigt im Vergleich von A_{out} zu $F_{2,out}$ eine Veränderung der Form des Eventclusters 3 (siehe Abbildung 10.6, Unterabbildung 1 zu 3). Die euklidische Distanz, die ein Maß der Veränderung darstellt, beträgt für diesen Testfall 0,39 (siehe Tabelle 12.3). Der Vergleich der Testoutput des KNNs (siehe Abbildung 10.6, Unterabbildung 4 zu 6) zeigt keine Identität, da die Anzahl der Eventcluster nicht übereinstimmt und im Ausgangstestfall A_{out} zwei Eventcluster dargestellt sind und im Folgetestfall $F_{2,out}$ drei Eventcluster.

10.2.3 $MMCB_{TL}1.3$ - Differenzieren zwischen Verhalten von Benutzern

Der Ausgangstestfall (siehe Abbildung 10.7) erfasst das Basisverhalten und illustriert die Fahrt von Benutzer A durch ein simuliertes Szenario, in dem zwei Events der Stufe 1 an unterschiedlichen Positionen, auftreten. Beide Events werden von Benutzer A initiiert. In den Folgetestfällen $F1, out$ und $F2, out$ hingegen wird eine Modifikation vorgenommen: die Events der Stufe 1 werden von zwei unterschiedlichen Benutzern ausgeführt, anstatt von einem Einzigen, wie es im Ausgangstestfall A_{out} der Fall ist.

Die $MMCB_{TL}1.3$ spezifiziert, wenn Benutzer A als Fahrer agiert, Eventcluster 1 im Folgetestfall $F1, out$ mit dem Eventcluster 1 im Ausgangstestfall A_{out} übereinstimmen muss. Ebenso muss der Eventcluster 2 identisch im Ausgangs- und Folgetestfall sein, falls Benutzer B der Fahrer ist. Diese Anforderung zielt darauf ab, die Fähigkeit des Systems zu überprüfen, zwischen verschiedenen Benutzern zu unterscheiden und ihr Verhalten korrekt zuzuordnen. Sollte das SuT in der Lage sein, die spezifischen Verhaltensweisen der einzelnen Benut-

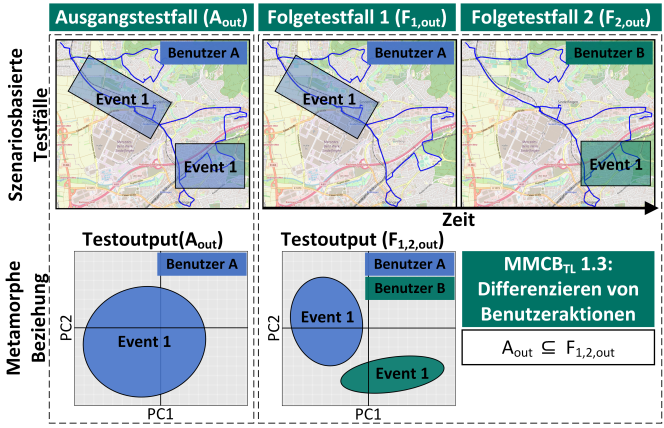

Abbildung 10.7: Beschreibung von Ausgangs- und Folgetestfälle für $MMCB_{TL}1.3$ für das Differenzieren zwischen Verhalten von Benutzern Benutzerverhalten

zer korrekt zu erkennen und zu trennen, wird die Anforderung $MMCB_{TL}1.3$ als erfüllt betrachtet. Wenn jedoch die Eventcluster der beiden Benutzer im Ausgangs- und Folgetestfall nicht übereinstimmen, zeigt dies, dass das System die Unterscheidung zwischen den Benutzerverhaltensweisen nicht umsetzen kann, und die MB gilt als nicht erfüllt. Diese Fähigkeit zur Differenzierung ist wesentlich für die Entwicklung von personalisierten Interaktionen innerhalb selbstlernender Systeme.

Evaluation der $MMCB_{TL}1.3$

Zur Evaluierung der $MMCB_{TL}1.3$ wird in diesem Testfall die Fensterfunktion betrachtet. Im Ausgangstestfall wird das Fenster an zwei geografischen Positionen geöffnet. Die daraus resultierende Eventcluster (Abbildung 10.8, siehe Unterabbildung 1) zeigt einen gemeinsamen Eventcluster für beide Fensteröffnungen.

In den Folgetestfällen $F_{1,out}$ und $F_{2,out}$ wird der Fensteröffnungen von zwei verschiedenen Benutzern ausgelöst. In Folgetestfall $F_{1,out}$ öffnet Benutzer A die das Fenster an der ersten Position, während in Folgetestfall $F_{2,out}$ Benutzer B das Fenster an der zweiten Position öffnet. Ein ML-Modell, das die Anforderungen der $MMCB_{TL}1.3$ erfüllt, muss in der Lage sein, diese unterschiedlichen

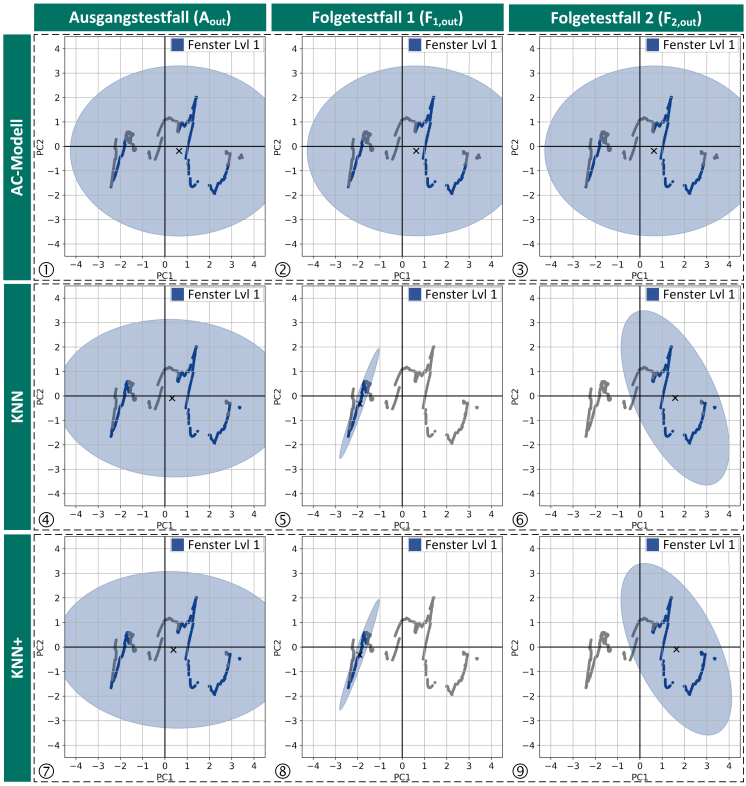

Abbildung 10.8: Evaluation der $MMCB_{TL}1.3$ für das Differenzieren von Benutzerverhalten

Benutzer zu erkennen und die Fensteröffnungen für den jeweiligen Benutzer individuell bereitzustellen.

Das AC-Modell zeigt in beiden Folgetestfällen $F_{1,out}$ und $F_{2,out}$ die identischen Eventcluster wie im Ausgangstestfall A_{out} und erfüllt somit die Anforderungen der $MMCB_{TL}1.3$ nicht. Die euklidische Distanz der Eventcluster beträgt für A_{out} und $F_{1,out}$ 0.034 und für A_{out} und $F_{2,out}$ 0.031[1]. Die beiden

[1] Bei der $MMCB_{TL}1.3$ weißt ein geringer euklidischer Abstand, im Gegensatz zu $MMCB_{TL}1.1$ und $MMCB_{ID}1.2$, auf das nicht-erfüllen der MB hin.

KNN-basierten ML-Modelle hingegen zeigen sowohl im Folgetestfall $F_{1,out}$ (Abbildung 10.8, siehe Unterabbildung 5 bzw. 6) als auch im Folgetestfall 2 $F_{2,out}$ (Abbildung 10.8, siehe Unterabbildung 8 bzw. 9) jeweils nur einen der beiden Fenstercluster, die eine Teilmenge des Ausgangstestfalls darstellen. Dies zeigt, dass die KNN-basierten Modelle die $\text{MMCB}_{\text{TL}}1.3$ erfüllen, indem sie die unterschiedlichen Benutzer und deren Fensteröffnungen korrekt erkennen und darstellen.

10.2.4 $\text{MMCB}_{\text{LR}}2.1$ - Modifikation des Zeitmerkmals

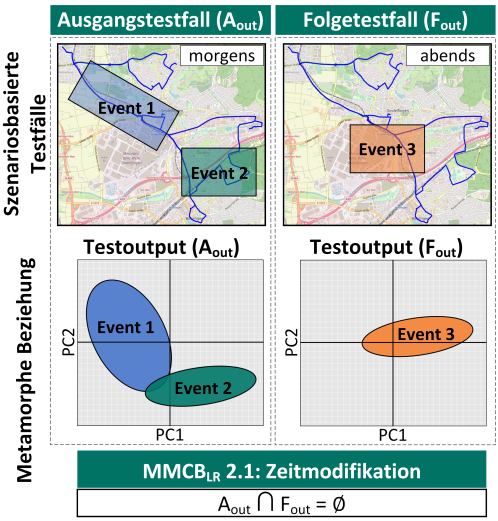

Abbildung 10.9: Beschreibung von Ausgangs- und Folgetestfall für $\text{MMCB}_{\text{LR}}2.1$ für die Modifikation des Zeitmerkmals

Eine Veränderung des Benutzerverhaltens kann, im Gegensatz zu den vorherigen MMCBs, bei der die Veränderung vom Benutzer selbst ausging, auch durch Änderungen in der Umwelt induziert werden. Die Umwelt hat einen unmittelbaren Einfluss auf das Verhalten des Benutzers und führt dadurch zu Veränderung in den Verhaltensweisen.

Die $\text{MMCB}_{\text{LR}}2.1$ untersucht die Beziehung von Testfällen in Bezug auf unterschiedliche Zeitpunkte. Der simulierte Benutzer des SSIB verändert sein

Verhalten demnach zu unterschiedlichen Zeitpunkten. In diesem Testaufbau wird ein gleichbleibendes Verhalten am Morgen angenommen (siehe Abbildung 10.9, Event 1 und Event 2). Abends hingegen wird ein anderes Verhalten ausgelöst, dargestellt durch Event 3. Die $MMCB_{LR}2.1$ beschreibt die Vereinigung des Testoutputs von A_{out} in Bezug auf F_{out} als leere Menge. Das bedeutet, dass kein Eventcluster 1 oder 2 im Testoutput F_{out} beobachtet werden darf. Falls ein anderer oder weiterer Eventcluster außer für Event 3 in F_{out} beobachtet wird, gilt die $MMCB_{LR}2.1$ als nicht erfüllt. Diese MB überprüft, ob das SuT in der Lage ist, zeitliche Muster korrekt zu erkennen und die entsprechenden Verhaltensänderungen zu adaptieren. Eine fehlerhafte Zuordnung der Eventcluster würde darauf hinweisen, dass das System nicht zwischen den zeitlich bedingten Verhaltensänderungen unterscheiden kann.

Evaluation der $MMCB_{LR}2.1$

Die Einhaltung der $MMCB_{LR}2.1$ wird anhand der Bewertung der Massagefunktion überprüft (siehe Abbildung 10.10). Die Evaluation umfasst zwei Szenarien. Im Ausgangstestfall A_{out} werden Benutzeraktionen der Intensitätsstufen Massage Level 1 und Level 2 in einem Zeitraum von 06:00 Uhr bis 08:00 Uhr erzeugt und über eine Dauer von 10 Tagen simuliert. Im Folgetestfall F_{out} wird das Massageevent Level 3 im Zeitraum von 17:00 Uhr bis 19:00 Uhr erzeugt, ebenfalls über eine Dauer von 10 Tagen. Bei der Betrachtung der Testausgaben des AC-Modells zeigt sich, dass weder im Ausgangstestfall A_{out} (siehe Abbildung 10.10, Unterabbildung 1) noch im Folgetestfall F_{out} (siehe Abbildung 10.10, Unterabbildung 2) Eventcluster beobachtbar sind. Das bedeutet, dass das AC-Modell keine Benutzeraktionen erlernen konnte und somit keine automatisierte Events erzeugt werden. Dieses Verhalten erfüllt nicht die durch die $MMCB_{LR}2.1$ gefordert Beziehung, die eine leere Clustermenge zwischen A_{out} und F_{out} vorschreibt.

Im Gegensatz dazu ergibt die Betrachtung des Ausgangstestfalls A_{out} (siehe Abbildung 10.10, Unterabbildung 3) des KNNs zwei Eventcluster für die Massagestufen Level 1 und Level 2. Im Folgetestfall F_{out} (siehe Abbildung 10.10, Unterabbildung 4) existiert lediglich ein Cluster für das Massageevent Level 3. Dieses Verhalten entspricht der in der $MMCB_{LR}2.1$ definierten Beziehung zwischen Ausgangs- und Folgetestfall. Die Testausgaben des KNN+ Modells (siehe Abbildung 10.10, Unterabbildung 5 bzw. 6) sind identisch mit denen des KNN-Modells und erfüllten somit ebenfalls die $MMCB_{LR}2.1$. Die Betrachtung

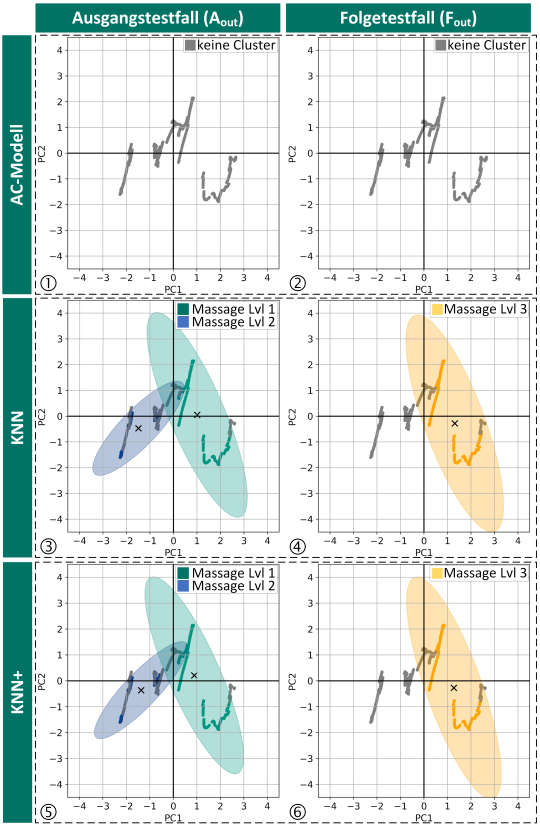

Abbildung 10.10: Evaluation der MMCB_{LR}2.1 für die Modifikation des Zeitmerkmals

des euklidischen Abstandes ist für diese Beziehung nicht anwendbar, da keine Eventclusterpaare zwischen A_{out} und F_{out} existieren und damit kein Abstand zwischen zwei Clusterzentren bestimmt werden kann. Vielmehr würde die Existenz eines euklidischen Abstandes darauf hinweisen, dass die MMCB_{LR} nicht erfüllt ist.

10.2.5 $MMCB_{TL}2.2$ - Erkennen von saisonalen Effekten

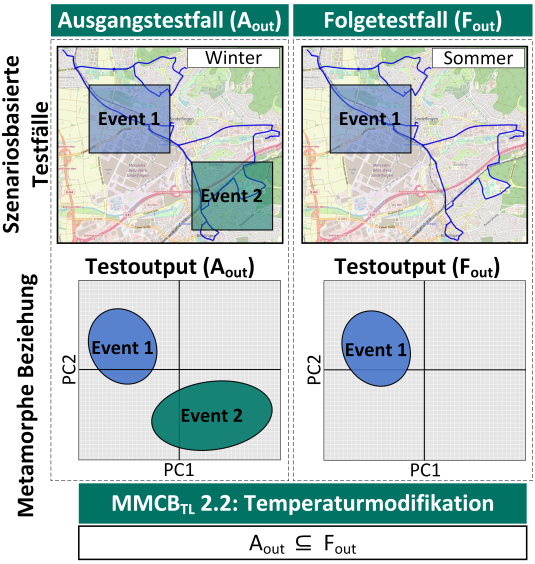

Abbildung 10.11: Beschreibung von Ausgangs- und Folgetestfall für $MMCB_{TL}2.2$ für das Erkennen von saisonalen Effekten

Die $MMCB_{TL}2.2$ (siehe Abbildung 10.11) betrachtet die Erkennung saisonaler Effekte und deren Einfluss auf das Verhalten von Fahrzeugnutzern. Saisonale Effekte, wie Jahreszeiten, können das Benutzerverhalten beeinflussen und damit zu unterschiedlichen Benutzeraktionen führen. Diese Variationen können auf saisonale Temperaturunterschiede zurückgeführt werden, wie sie zwischen Sommer- und Wintermonaten auftreten. Andere wetterbedingte Faktoren wie Regen, Schneefall oder die Intensität der Sonneneinstrahlung werden in dieser Untersuchung nicht berücksichtigt. Im Ausgangstestfall A_{out} wird ein Szenario während der Wintermonate beschrieben. Dieser Kontext führt zur Aktivierung zweier Events, welche unter anderem die Automatisierung der Sitzheizung initiieren.

Der Folgetestfall F_{out} findet in den Sommermonaten statt, in denen aufgrund der höheren Temperaturen die Benutzeraktionen des Fahrers variieren. In die-

sem Szenario wird Event 1 ausgelöst, welches an der gleichen Position wie das Event 2 im A_{out} ausgeführt wird.

Die $MMCB_{TL}2.2$ definiert eine Beziehung zwischen den Testergebnissen des Ausgangs- und des Folgetestfalls, die zeigt, dass eine Teilmenge zwischen den Eventcluster für Stufe 1 vorliegt. Diese Analyse betont die Bedeutung des Situationsbewusstseins und der Anpassungsfähigkeit von selbstlernenden Systemen im Kontext variierender Umgebungsbedingungen.

Evaluation der $MMCB_{TL}2.2$

Die Evaluation der $MMCB_{TL}2.2$ wird anhand der Sitzheizungsfunktion durchgeführt. Sitzheizung Level 1 (siehe Abbildung 10.12) entspricht der niedrigsten Stufe der Sitzheizung, während Sitzheizung Level 2 einer höheren Stufe entspricht. Das Szenario wurde so gewählt, dass der Ausgangstestfall A_{out} für 30 Tagen der Wintermonaten (Dezember-Februar) simuliert wurde. Der Folgetestfall F_{out} wurde anhand 30 Tagen in den Sommermonaten (Juni-August) simuliert, die höhere Durchschnittstemperaturen aufweisen und daher nur Aktivierungen der Sitzheizung auf Stufe 1 auftreten.

Das AC-Modell zeigt einen identischen Eventcluster der Sitzheizung Stufe 1 sowohl im Ausgangstestfall A_{out} als auch im Folgetestfall F_{out}. Das beobachtete Verhalten des AC-Modells, das identische Eventcluster im Ausgangs- und Folgetestfall zeigt, erfüllt daher die $MMCB_{TL}2.2$ nicht. Der euklidische Abstand zwischen den Clusterzentren der Eventcluster im Ausgangstestfall A_{out} und Folgetestfall F_{out} beträgt 0.008, wodurch die Identität bestätigt wird (siehe Tabelle 12.3).

Die Analyse der Testoutputs der KNN-basierten Modelle zeigt, dass zwei Cluster für die Sitzheizung Stufe 1 und Stufe 2 im Ausgangstestfall A_{out} erzeugt wurden (siehe Abbildung 10.12, Unterabbildung 3 bzw. 4). Im Folgetestfall F_{out} ist kein Cluster für Sitzheizung Stufe 2 erkennbar und die Dimensionen des Clusters für Sitzheizung Stufe 1 sind reduziert (siehe Abbildung 10.12, Unterabbildung 5 bzw. 6). Damit erfüllen die KNN-basierten Modelle die $MMCB_{TL}2.2$, die vom Typ der Teilmenge ist. Diese Ergebnisse verdeutlichen, dass die KNN-basierten Modelle robuster gegenüber saisonalen Variationen sind und daher die Anforderungen der $MMCB_{TL}2.2$ erfüllen als das AC-Modell.

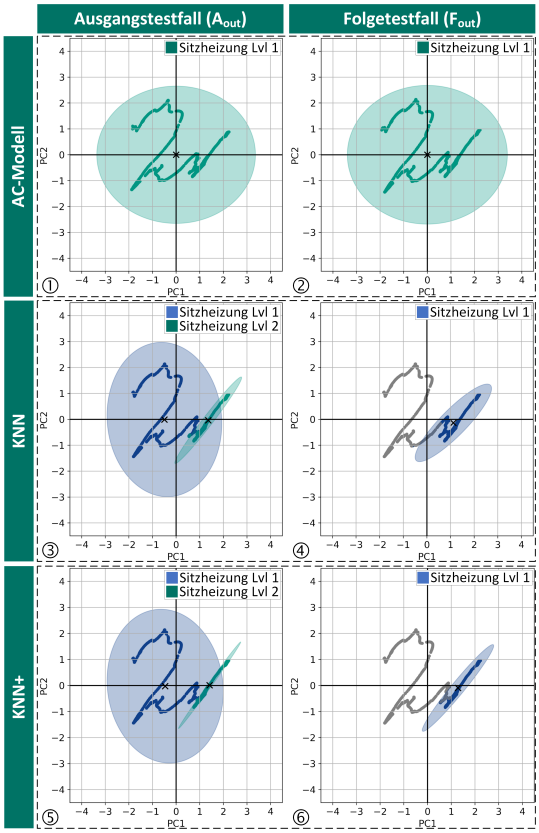

Abbildung 10.12: Evaluation der $MMCB_{TL}2.2$ für das Erkennen von saisonalen Effekten

10.2.6 $MMCB_{TL}2.3$ - Modifikation der Fahrtrichtung

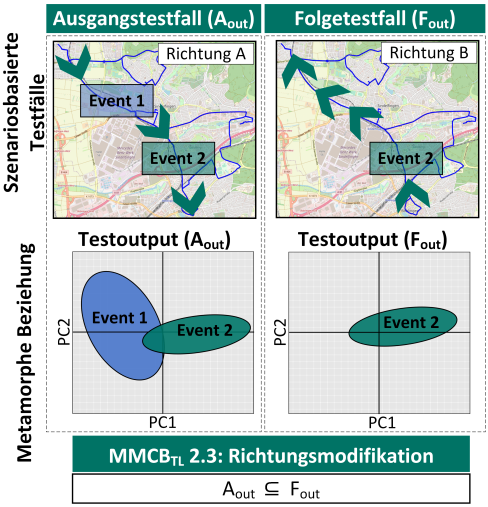

Abbildung 10.13: Beschreibung von Ausgangs- und Folgetestfall für $MMCB_{TL}2.3$ für die Modifikation der Fahrtrichtung

Die $MMCB_{TL}2.3$ untersucht ebenfalls das Situationsbewusstsein der ML-Modelle, welche die Fähigkeit des Systems umfasst, den aktuellen Kontext zu erkennen. Die Eigenschaft $MMCB_{TL}2.3$ untersucht Richtungsänderungen in einem gegebenen Szenario (siehe Abbildung 10.13). Insbesondere für personalisierte Komfortfunktionen ist die Kenntnis der Fahrtrichtung von Bedeutung. Ein Benutzer könnten auf dem Hinweg andere Benutzeraktionen wählen als auf dem Rückweg, was eine differenzierte Erkennung und Reaktion seitens des SSIB erfordert.

Der Ausgangstestfall A_{out} definiert eine spezifische Richtung, genannt „Richtung A“, in der ein Event 1 und Event 2, simuliert wird. Im Folgetestfall F_{out} wird die identische Strecke in entgegengesetzter Richtung befahren. Im Gegensatz zu dem Ausgangstestfall A_{out} wird im Folgetestfall F_{out} das Event 2 ausgelöst. Wenn nun im Folgetestfall F_{out} simuliert wird, darf nur das Event 2, welches identisch zum Ausgangstestfall A_{out} sein muss, festgestellt werden. Das Event 1 darf im Folgetestfall F_{out} nicht vertreten sein. Die $MMCB_{TL}2.3$ fordert, somit dass zwischen den Testoutputs des Ausgangsfalls A_{out} und des

Folgetestfalls F_{out} eine Teilmenge festgestellt werden kann (siehe Unterabschnitt 9.1.1). Dies unterstreicht die Notwendigkeit, dass das System unterschiedliche Richtungen erkennt und adäquat auf die damit verbundenen unterschiedlichen Benutzeraktionen reagiert.

Evaluation der $MMCB_{TL}2.3$

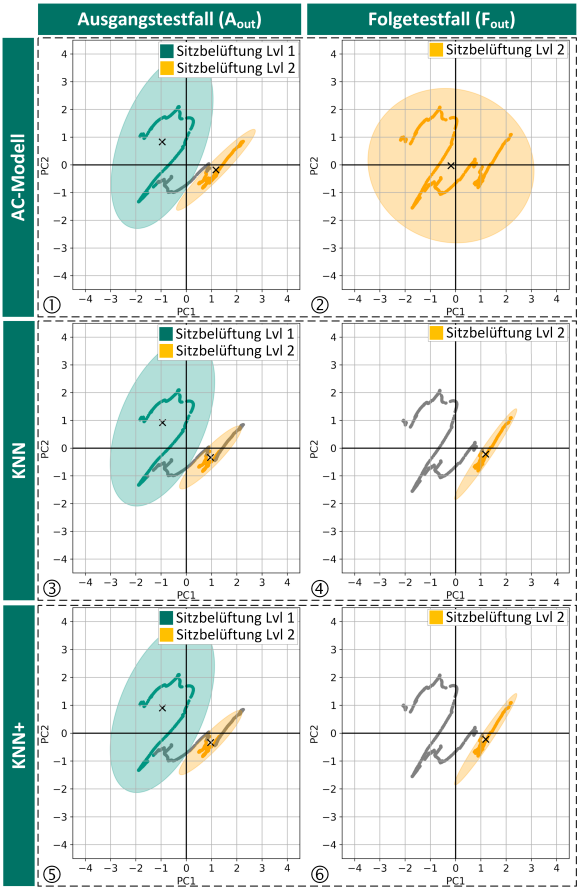

Abbildung 10.14: Evaluation der $MMCB_{TL}2.3$ für die Modifikation der Fahrtrichtung

Die Evaluation der $MMCB_{TL}2.3$ wird am Beispiel der Sitzbelüftungsfunktion durchgeführt (siehe Abbildung 10.14). Im Ausgangstestfall A_{out} wird das Testszenario in Richtung A mit den Sitzbelüftung 1 und 2 befahren. Im Folgetestfall F_{out} wird das Testszenario in umgekehrter Richtung und mit der Sitzbelüftung der Stufe 2 simuliert. Bei der Testdurchführung müssen somit im Ausgangstestfall A_{out} und im Folgetestfall F_{out} identische Eventcluster für Sitzbelüftung 2 visualisiert werden. Das Eventcluster für die Sitzheizung der Stufe 1, darf nicht mehr im Folgetestfall F_{out} sichtbar sein. Der Vergleich des Ausgangstestfalls A_{out} des AC-Modell (Abbildung 10.14, Unterabbildung 1) mit dem Folgetestfall F_{out} (Abbildung 10.14, Unterabbildung 2) zeigt keine identischen Eventcluster für die Sitzbelüftung Stufe 2. Der Eventcluster im Folgetestfall ist vergrößert und es werden alle Datenpunkte der Sitzbelüftung der Stufe 2 zugeordnet. Dadurch verschieben sich die Clusterzentren und der euklidische Abstand der beiden Eventcluster beträgt 1.30. Die visuelle Betrachtung und der Wert des euklidischen Abstandes zeigen, dass die $MMCB_{TL}2.3$ durch das AC-Modell nicht erfüllt ist.

Das KNN und das KNN+ Modell zeigen einen Eventcluster für die Sitzbelüftung Stufe 2, die eine Teilmenge im Vergleich der Ausgangstestfälle A_{out} (Abbildung 10.14, Unterabbildung 3,5) zu den Folgetestfällen F_{out} (Abbildung 10.14, Unterabbildung 4,6) darstellt. Der euklidische Abstand der Eventcluster beträgt für das KNN-Modell 0.001 und für das KNN+ Modell 0.008 (siehe Tabelle 12.3). Durch diese Identität erfüllen die beiden KNN-basierten ML-Modelle die $MMCB_{TL}2.3$ und sind somit in der Lage, die Richtung des simulierten Szenarios zu erfassen.

10.2.7 $MMCB_{ID}3.1$ - Funktionsfähigkeit mit Messrauschen

Das Ziel der $MMCB_{ID}3.1$ besteht darin, die Robustheit des SuT hinsichtlich Sensorrauschen zu bewerten (siehe Abbildung 10.15). Sensorrauschen resultiert aus technischen Beschränkungen der Sensoren, die es verhindern, perfekte Messwerte zu liefern. Die Spezifität des Rauschens variiert je nach Sensortyp. Insbesondere das GPS-Signal ist aufgrund von Umgebungseinflüssen wie Abschattungen einem Rauschen von bis zu 5 Metern ausgesetzt. Obwohl alle Signale von Rauschen betroffen sein können, konzentriert sich diese MMCB auf die Überprüfung der Systemrobustheit gegenüber solchem Rauschen. Der Ausgangstestfall simuliert ein Szenario, in dem das GPS-Signal,

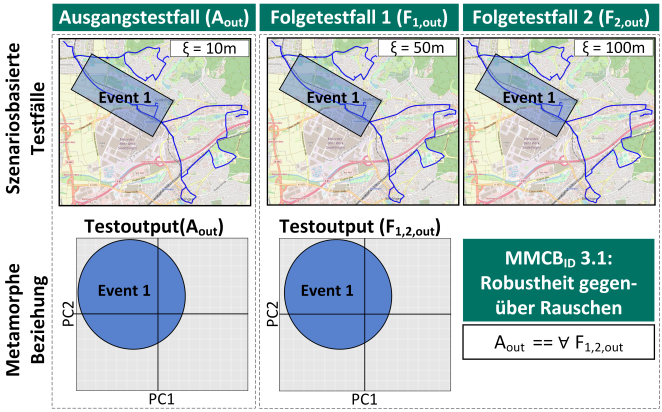

Abbildung 10.15: Beschreibung von Ausgangs- und Folgetestfällen von MMCB$_{ID}$3.1 für die Funktionsfähigkeit mit Messrauschen

bestehend aus Breiten- und Längengrad, frei von Rauschen ist. Die nachfolgenden Testfälle replizieren dieses Szenario mit einer schrittweisen Erhöhung des GPS-Rauschens. Die MMCB$_{ID}$3.1 beschreibt die Beziehung zwischen dem ursprünglichen Testoutput A_{out} und den Ergebnissen der Folgetestfälle $F_{1,out}$ und $F_{2,out}$. Diese MMCB wird als Identitätsbeziehung definiert, wobei erwartet wird, dass die Ausgabewerte identisch sind oder innerhalb eines vorab definierten Identitätskriteriums liegen, wie in Gleichung 9.1 dargestellt.

Evaluation der MMCB$_{ID}$3.1

Die Evaluation der MMCB$_{ID}$3.1 (siehe Abbildung 10.16) erfolgt anhand der Fenster-Komfortfunktion, die Fensteröffnungen erlernt. Ausgegraute Datenpunkte deuten darauf hin, dass an diesen Stellen keine Aktivierung durch das ML-Modell erfolgt. Die Datenpunkte des Ausgangstestfalls werden mit einem Perlin-Rauschen (siehe Abbildung 8.6) überlagert, wodurch ein Rauschen in den Latitude- und Longitude-Merkmalen erzeugt wird.

In den Folgetestfällen $F_{1,out}$ und $F_{2,out}$ wird die Stärke des Rauschens für die Latitude und Longitude auf 100 Meter bzw. 150 Meter erhöht. Zu Beginn zeigen alle drei untersuchten ML-Modelle ein ähnliches Verhalten (MMCB$_{ID}$3.1, Unterabbildung 1, 4, 7). Der Folgetestfall $F_{1,out}$ des AC-

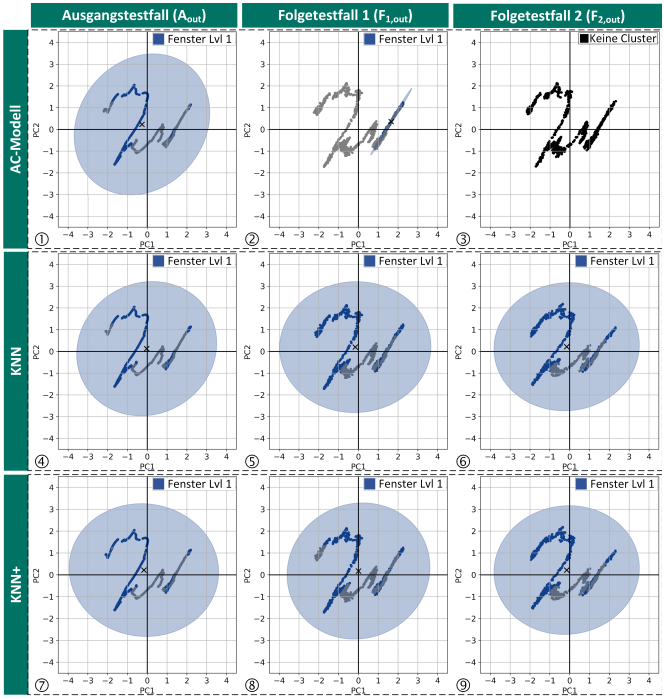

Abbildung 10.16: Evaluation der $MMCB_{ID}3.1$ für die Funktionsfähigkeit mit Messrauschen

Modells ($MMCB_{ID}3.1$, Unterabbildung 2) zeigt im Vergleich zum Ausgangstestfall ($MMCB_{ID}3.1$, Unterabbildung 1) eine Reduktion des Eventclusters und somit eine Reduktion der Anzahl der Fensteröffnungen. Das Clusterzentrum verschiebt sich und der euklidische Abstand zwischen A_{out} und $F_{1,out}$ beträgt 1.18. Dadurch wird bereits im Folgetestfall $F_{1,out}$ die $MMCB_{ID}3.1$, die ein identisches Verhalten über alle Folgetestfälle fordert, nicht erfüllt. Diese Beobachtung wird durch den Vergleich des Ausgangstestfalls A_{out} ($MMCB_{ID}3.1$, Unterabbildung 1) mit dem Folgetestfall $F_{2,out}$ ($MMCB_{ID}3.1$, Unterabbildung 3) weiter bestätigt. In $F_{2,out}$ des AC-Modells ($MMCB_{ID}3.1$, Unterabbildung 3) ist kein Eventcluster vorhanden, was bedeutet, dass keine Eventaktivierung durch das ML-Modell erfolgt und somit die $MMCB_{ID}3.1$ ebenfalls nicht erfüllt ist.

Im Gegensatz dazu erfüllen beide KNN-basierten ML-Modelle die $MMCB_{ID}3.1$. Die Eventcluster für Fensteraktivierungen bleiben beim Vergleich des Ausgangstestfalls A_{out} ($MMCB_{ID}3.1$, Unterabbildung 4 bzw. 7) mit dem Folgetestfall $F_{1,out}$ ($MMCB_{ID}3.1$, Unterabbildung 5 bzw. 8) und Folgetestfall $F_{2,out}$ ($MMCB_{ID}3.1$, Unterabbildung 6 bzw. 9) identisch (siehe Tabelle 12.3). Dies zeigt, dass die KNN-basierten Modelle robust gegenüber dem erhöhten Rauschen sind und die Anforderungen der $MMCB_{ID}3.1$ erfüllen.

10.2.8 $MMCB_{ID}3.2$ - Ignorieren von Ausreißern

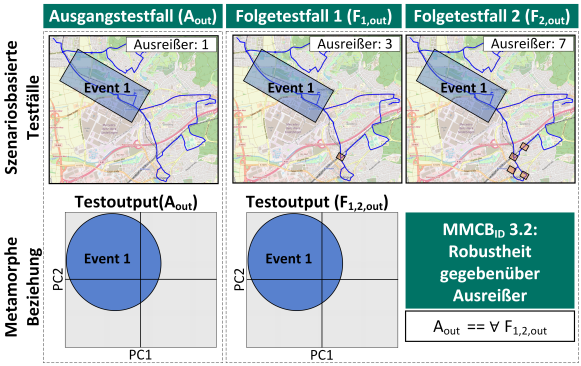

Abbildung 10.17: Beschreibung von Ausgangs- und Folgetestfall für $MMCB_{ID}3.2$ für das Ignorieren von Ausreißern

Der zweite Teilaspekt der Robustheitseigenschaft, dargelegt in Tabelle 9.1, fokussiert sich auf den Umgang mit Ausreißern. In diesem Zusammenhang bezeichnet ein Ausreißer eine ungewollte Aktivierung eines Ereignisses. Während vereinzelte solche Aktivierungen unerwünschte Störungen darstellen, kann eine Ansammlung derselben auf ein potenziell neues, zu adaptierendes Verhaltensmuster hinweisen. Es ist daher erforderlich, das SSIB so zu parametrisieren, dass nicht jede einzelne Aktivierung einer Komfortfunktion fälschlicherweise als Ereigniscluster interpretiert wird. Gleichzeitig darf die Sensibilität nicht derart reduziert werden, dass ein neues Ereignis erst nach einer unverhältnismäßig hohen Anzahl von Aktivierungen erkannt wird. Die optimale Einstellung der Parameter, um dieses Gleichgewicht zu erreichen, obliegt dem Entwickler und wird durch diesen Teilaspekt einer Überprüfung

unterzogen. Der Ausgangstestfall simuliert ein Szenario ohne jegliche Ausreißer (siehe Abbildung 10.17). In den folgenden Testfällen $F_{1,out}$ und $F_{2,out}$ wird die Anzahl der Ausreißer schrittweise erhöht. Es wird erwartet, dass die Beziehung zwischen den Testergebnissen A_{out} und $F_{1,out}$ und $F_{2,out}$ konsistent bleibt, das heißt, dass keine neuen Verhaltensweisen innerhalb der Testsequenzen erlernt und zufällige Aktivierungen ignoriert werden.

Evaluation der $MMCB_{ID}3.2$

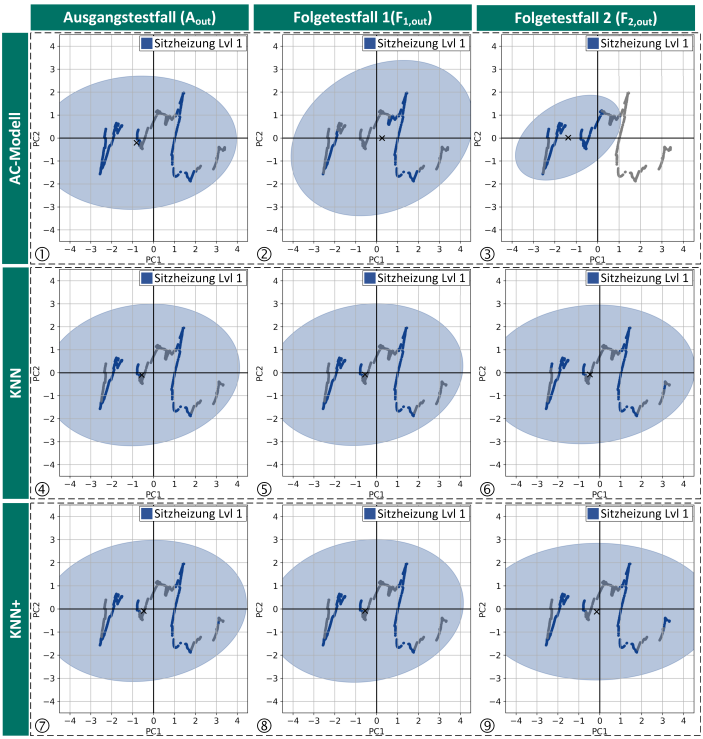

Abbildung 10.18: Evaluation der Ergebnisse $MMCB_{ID}3.2$ für das Ignorieren von Ausreißern

Der Testfall $MMCB_{ID}3.2$ (siehe Abbildung 10.18) untersucht die Benutzeraktionen mit der Sitzheizungsfunktion, die vier Zustände aufweist: Level 0

(aus) und Level 1-3 (steigende Intensitätsstufen). Dieser Testfall ist analog zu $MMCB_{ID}3.1$ aufgebaut, jedoch liegt der Unterschied in der untersuchten Teileigenschaft „Ausreißer“. In jedem Folgetestfall F_{out} wird die Anzahl der Ausreißer sukzessiv erhöht. Der Ausgangstestfall A_{out} weist keine Ausreißer auf, während Folgetestfall $F_{1,out}$ drei Ausreißer und Folgetestfall $F_{2,out}$ fünf Ausreißer der Benutzeraktion „Sitzheizung“ aufweisen.

Für das Erfüllen der $MMCB_{ID}3.2$ müssen die Eventcluster der Sitzheizung im Ausgangstestfall A_{out} und allen Folgetestfällen $F_{1,2,out}$ identisch sein. Der Vergleich des Testoutputs des AC-Modells des Ausgangstestfalls A_{out} (Abbildung 10.18, Unterabbildung 1) mit dem von Folgetestfall $F_{1,out}$ (Abbildung 10.18, Unterabbildung 2) zeigt eine Veränderung des Eventclusters. Diese Veränderung manifestiert sich durch die veränderte Form des Clusters und die damit verbundene Verschiebung des Clustermittelpunktes. Die Veränderungen setzen sich im nachfolgenden Testfall $F_{2,out}$ (Abbildung 10.18, Unterabbildung 3) fort, wobei der Ereigniscluster im Vergleich zum Ausgangstestfall A_{out} abweicht. Die Abweichung der euklidischen Distanz zwischen A_{out} und $F_{1,out}$ beträgt 0.225 und zwischen A_{out} und $F_{2,out}$ 0,430 (siehe Tabelle 12.3). Damit wird die $MMCB_{ID}3.2$, die ein identisches Verhalten über alle Folgetestfälle im Vergleich zum Ausgangstestfall voraussetzt, nicht erfüllt. Das AC-Modell zeigt ein Verhalten, das sich mit der Anzahl der Ausreisen verändert.

Die Ergebnisse der KNN- und KNN+ Modelle erfüllen die Anforderungen der $MMCB_{ID}3.2$. Die Ereigniscluster der Sitzheizung bleiben im Ausgangstestfall A_{out} (Unterabbildung 4 bzw. Unterabbildung 7), im Folgetestfall $F_{1,out}$ (Unterabbildung 5 bzw. Unterabbildung 8) und im Folgetestfall $F_{2,out}$ (Unterabbildung 6 bzw. Unterabbildung 9) unverändert, was den Vorgaben der $MMCB_{ID}3.2$ entspricht. Zusammenfassend zeigt sich, dass die $MMCB_{ID}3.2$ durch die KNN- und KNN+ Modelle erfüllt wird, während das AC-Modell bei zunehmender Anzahl von Ausreißern Abweichungen im Ereigniscluster aufweist.

10.3 Zusammenfassung und Diskussion der Ergebnisse

Die Evaluation der SSIB durch die drei ML-Modelle im Testszenario „Arbeit“ zeigt, dass das AC-Modell ausschließlich der $MMCB_{TL}1.1$ genügt, wohinge-

Tabelle 10.1: Übersicht der erfüllten (✓) und fehlgeschlagenen (✗) MMCBs anhand der drei untersuchten Szenarien

ID	Arbeit			Erledigung			Begleitung		
	AC	KNN	KNN+	AC	KNN	KNN+	AC	KNN	KNN+
Differenzierungs-fähigkeit									
$MMCB_{TL}1.1$	✓	✓	✓	✓	✓	✓	✓	✓	✓
$MMCB_{ID}1.2$	✗	✗	✓	✗	✗	✓	✗	✗	✓
$MMCB_{TL}1.3$	✗	✓	✓	✗	✓	✓	✗	✓	✓
Umgebungs-verhalten									
$MMCB_{LR}2.1$	✗	✓	✓	✗	✓	✓	✗	✓	✓
$MMCB_{TL}2.2$	✗	✓	✓	✗	✓	✓	✗	✓	✓
$MMCB_{TL}2.3$	✗	✓	✓	✗	✓	✓	✗	✓	✓
Robustheit									
$MMCB_{ID}3.1$	✗	✓	✓	✗	✓	✓	✗	✓	✓
$MMCB_{ID}3.2$	✗	✓	✓	✗	✓	✓	✗	✓	✓
Erfüllte MMCBs	**1/8**	**7/8**	**8/8**	**1/8**	**7/8**	**8/8**	**1/8**	**7/8**	**8/8**

gen es bei allen weiteren MMCBs diese nicht erfüllt. Die $MMCB_{TL}1.1$ bewertet den grundlegenden Aspekt des Erlernens von Benutzeraktionen. Dieser Aspekt lässt sich auch durch die Verwendung einzelner Testfälle des szenariobasierten Testens (siehe Unterabschnitt 2.3.1) validieren. Es ist anzunehmen, dass während der Entwicklung des AC-Modells eine szenariobasierte Validierung erfolgte, was dessen Erfüllung der $MMCB_{TL}1.1$ erklären könnte. Alle weiteren MMCBs (siehe Tabelle 9.1) betrachten jedoch nicht das Erfüllen eines einzelnen Testfalls, sondern setzen mehrere Testfälle in eine metamorphe Beziehung zueinander. Diese metamorphen Beziehungen ermöglichen es, Fehler im Verhalten des SuT zu identifizieren, die über die Einzelbetrachtung von Testfällen hinausgehen.

Das KNN-Modell erfüllt sieben der acht definierten MMCBs. Die $MMCB_{ID}1.2$, die den Aspekt des Vergessens von Benutzeraktionen bewertet, wird in keinem der Testszenarien durch das erfüllt. Aus dieser Beobachtung wird geschlossen, dass das KNN-Modell einmal gelernte Benutzeraktionen nicht wieder vergessen kann und somit die Anzahl der erlernten Benutzeraktionen immer weiter erhöht, wodurch das SuT nicht mehr nutzbar sein wird. Das KNN+ Modell, welches mit einer Logik zum erneuten Training ausgestattet ist, erfüllt alle

MMCBs und erfüllt somit die geforderten Teil-Aspekte und Eigenschaften (siehe Tabelle 9.1).

Die Ergebnisse sind auch in den Testszenarien „Erledigung“ und „Begleitung“ zu beobachten, in denen das AC-Modell ebenfalls nur die $MMCB_{TL}1.1$ erfüllt. Diese konsistenten Ergebnisse verdeutlichen, dass die Evaluation unabhängig vom gewählten Szenario ist und das AC-Modell in allen getesteten Szenarien eine geringe Performanz aufweist (siehe Tabelle 10.1).

Der Vorteil des MT lässt sich exemplarisch an der $MMCB_{TL}2.3$ verdeutlichen. Diese MMCB gehört zum Muster der Identität, wodurch das identische Eventcluster im Testoutput des Ausgangstests und im nachfolgenden Testfall sichtbar sein müssen. Beim AC-Modell offenbart der Vergleich zwischen dem Testoutput des Ausgangstestfalls (siehe Abbildung 10.14, Unterabbildung 1) und dem Folgetestfall einen hohen euklidischen Abstand von 1.30 in den Eventclustern. Entscheidend ist hierbei nicht die isolierte Form oder Position der Eventcluster in den einzelnen Testfällen, sondern ihre gegenseitige Beziehung, wie sie durch $MMCB_{TL}2.3$ festgelegt ist. Dies unterstreicht, dass Metamorphes Testen primär auf die Konsistenz von Relationen zwischen den Testausgaben abzielt, anstatt auf die einzelnen Ausgaben per se. Dadurch kann gefolgert werden, ob ein Fehlverhalten im SuT vorliegt, ohne das korrekte Verhalten im einzelnen Testfall zu kennen.

Auffallend ist zudem, dass die Testoutputs der Ausgangstestfälle der ML-Modelle für MMCB 2.2_{LR}. Die Ausgangstestfälle des AC-Modells (Abbildung 10.12, Unterabbildung 1) und der KNN-basierten ML-Modelle (Abbildung 10.12, Unterabbildung 3 und Unterabbildung 5) unterscheiden, obwohl den ML-Modellen identische Trainings und Testdaten zur Verfügung gestellt wurden. Das unterschiedliche Verhalten bereits in den Ausgangstestfällen lässt sich durch die Betrachtung der Genauigkeit erklären. Das AC-Modell weist im Ausgangstestfall eine Genauigkeit von 56,13%. Genauigkeit (siehe Unterabschnitt 2.2.3) beschreibt den Anteil der Datenpunkte, die dem korrekten Zustand der Sitzheizung zugeordnet wurden, im Verhältnis zur Gesamtanzahl der Datenpunkte. Der Folgetestfall des AC-Modells besitzt eine Genauigkeit von 43.86%. Im Gegensatz dazu zeigt das KNN-Modell im Ausgangstestfall (Abbildung 10.12, Unterabbildung 3) eine Genauigkeit von 100% und im Folgetestfall (Abbildung 10.12, Unterabbildung 4) eine Genauigkeit von 90.6%. Das KNN+ Modell weist ebenfalls hohe Genauigkeitswerte auf: im Ausgangstestfall (Abbildung 10.12, Unterabbildung 5) eine Genauigkeit von 96% und

im Folgetestfall (Abbildung 10.12, Unterabbildung 6) eine Genauigkeit von 93%. Durch die Evaluation von drei unterschiedlichen ML-Modellen ist nicht nur ein Vergleich der Ausgangs- und Folgetestfälle möglich, sondern auch ein Vergleich der Testergebnisse aller drei ML-Modelle untereinander sowie eine zusätzliche Evaluierung im Sinne des Differenziellen Testens (siehe Unterabschnitt 4.1.1). Des weiteren können die Genauigkeiten der Testfälle des Szenarios „Arbeit“ einzeln betrachtet (siehe Tabelle 10.2) werden. Dieses Vorgehen entspricht einem traditionellen szenariobasierten Testansatz, bei dem Testszenarien als Eingaben definiert und die Testergebnisse mit den zuvor festgelegten erwarteten Ergebnissen verglichen werden. Dieses Vorgehen kann angewandt werden, da durch CAGEN Benutzeraktionen Abschnitt 8.3 synthetisch erzeugt wurden und dadurch ein „korrektes“ Verhalten definiert wurde.

Der Vergleich der erfüllten MMCBs (Tabelle 10.1) mit der Tabelle der Genauigkeiten (Tabelle 10.2) zeigt den Vorteil des Metamorphen Testens. Beispielsweise zeigt sich bei MMCB 1.3_{LR}, dass die Genauigkeit im Ausgangstestfall A_{out} 90,5% und im Folgetestfall F_{2out} 89,2% beträgt. Die hohen Genauigkeitswerte suggerieren, dass das AC-Modell die Testfälle erfüllt. Allerdings zeigt die Evaluation dieser Testfälle anhand der MMCB 1.3_{ID}, dass die Metamorphe Relationen nicht erfüllt ist und obwohl Genauigkeitswerte von ca. 90% erreicht wurde, kein korrektes Verhalten vorliegt. Die Limitationen der Testmethode können anhand der Ergebnisse der MMCB 2.1_{LR} analysiert werden. Diese spezifische MMCB fordert eine leere Schnittmenge zwischen den Ereignisclustern des Ausgangs- und des Folgetestfalls. Bei der Betrachtung von Eventcluster des AC-Modells von $\mathrm{MMCB}_{\mathrm{LR}}2.1$ (Abbildung 10.10, Unterabbildung 1 und 2) ist keine Schnittmenge der Eventcluster erkennbar und somit formal die MMCB erfüllt (siehe Gleichung 9.1). Allerdings tritt dieser Fall nur ein, weil das AC-Modell keinerlei Eventcluster extrahiert hat und diese somit eine leere Menge bilden. Ein ähnliches Problem kann auch bei MMCBs vom Typ der Identität auftreten, bei denen mehrere leere Eventcluster fälschlicherweise als identisch betrachtet werden, obwohl lediglich keine Eventcluster extrahiert wurden. Dieser Fall ist in den Untersuchten MMCBs nicht aufgetreten, wäre aber denkbar. Um dieser Limitation zu begegnen, kann eine Anpassung der Definition der MMCBs erwogen werden. Eine solche Anpassung könnte festlegen, dass eine MMCB nur dann als erfüllt gilt, wenn tatsächlich Eventcluster extrahiert wurden. Diese Änderung könnte jedoch die Anwendbarkeit der Testmethode einschränken, insbesondere wenn das Untersuchungsziel gerade darin besteht, dass keine Eventcluster gebildet werden.

Tabelle 10.2: Tabelle der Genauigkeiten für Ausgangstestfall und Folgetestfälle im Vergleich zu den Ergebnissen der MMCBs (nach Tabelle 10.1) für Szenario „Arbeit“

	ML-Modell	**Ausgangstestfall** (A_{out})	**Folgetestfall** ($F_{1,out}$)	**Folgetestfall** ($F_{2,out}$)	**Szenario „Arbeit“ (nach Tabelle 10.1)**
$MMCB_{TL}$1.1	AC	0.8970	0.8970	0.7625	✓
	KNN	0.9976	0.9832	0.9896	✓
	KNN+	0.9992	0.9856	0.9880	✓
$MMCB_{ID}$1.2	AC	0.8333	0.5951	0.8333	✗
	KNN	0.8968	0.8641	0.7721	✗
	KNN+	0.9184	0.8545	0.9184	✓
$MMCB_{TL}$1.3	AC	0.9056	0.5597	0.8922	✗
	KNN	0.9408	0.9976	0.9336	✓
	KNN+	0.9400	0.9976	0.9192	✓
$MMCB_{LR}$2.1	AC	0.4504	0.7539	-	✗
	KNN	0.9215	0.9141	-	✓
	KNN+	0.9303	0.9157	-	✓
$MMCB_{TL}$2.2	AC	0.5613	0.4386	-	✗
	KNN	1.0	0.9064	-	✓
	KNN+	0.9736	0.9408	-	✓
$MMCB_{TL}$2.3	AC	0.8970	0.2712	0.8922	✗
	KNN	0.9976	0.9664	0.9960	✓
	KNN+	0.9984	0.9888	0.9944	✓
$MMCB_{ID}$3.1	AC	0.0	0.0	0.0	✗
	KNN	0.9680	0.9616	0.8121	✓
	KNN+	0.9240	0.9496	0.8033	✓
$MMCB_{ID}$3.2	AC	0.8380	0.8380	0.6823	✗
	KNN	0.9776	0.9792	0.9848	✓
	KNN+	0.9912	0.9776	0.9880	✓

Eine alternative Lösung könnte darin bestehen, zusätzlich zur Analyse der MMCB-Erfüllung die Genauigkeit der den MMCBs zugeordneten Testfälle zu evaluieren (siehe Tabelle 10.2). Testfälle mit einer Genauigkeit unter einem bestimmten Schwellenwert (etwa unter 75%) könnten als nicht erfüllt gekennzeichnet werden. Diese Maßnahme würde eine differenziertere Bewertung der Testergebnisse ermöglichen und sicherstellen, dass die Testauswertungen die tatsächliche Leistung des ML-Modells widerspiegeln. Dadurch kann die Kombination von szenariobasiertem Testen mit metamorphem Testen erfolgen. Während das szenariobasierte Testen eine erste Prüfung darstellt, indem es bestätigt, ob bestimmte Szenarien erfolgreich durchlaufen werden oder nicht,

ermöglicht das metamorphe Testen eine tiefere Analyse der Testergebnisse. Durch die Identifikation spezifischer Eigenschaften oder Anforderungen, die verletzt wurden, liefert das metamorphe Testen zusätzlich Einblicke, welche Eigenschaften und Teilaspekte nicht erfüllt sind. Diese Kombination verbessert damit die Fehlererkennung und -diagnose, da sie nicht nur feststellt, dass ein Problem existiert, sondern auch aufzeigt, welche spezifischen Bedingungen oder Annahmen verletzt wurden. Somit bietet die kombinierte Methodik eine umfassendere Validierung von Software, erhöht die Zuverlässigkeit der Testergebnisse und unterstützt eine gezieltere Fehlerbehebung.

11 Zusammenfassung und Ausblick

11.1 Zusammenfassung

Infotainmentsysteme in Fahrzeugen repräsentieren eine auf Komfortsysteme spezialisierte SSIB, die Benutzeraktionen in einem Kontext aufzeichnet, daraus individuelle Muster generalisiert und entsprechend die Aktionen automatisiert. Mit den stetigen Fortschritten der KI und der zunehmenden Kooperation zwischen Menschen und KI-Systemen ist es unabdingbar, die Ergebnisse von SSIB zu validieren, um somit Vertrauen in diese zu generieren.

Bislang sind die Methoden und Werkzeuge zur Validierung eines KI-Systems jedoch unzureichend und müssen an die neuen Arten von selbstlernenden Algorithmen angepasst werden. Traditionelle Testmethoden basieren auf der Erstellung von Testfällen und der Existenz eines Testorakels.

Gegenwärtig stellt die Erstellung eines Testorakels für selbstlernende Systeme eine Herausforderung dar, da gewünschte Eigenschaften formal schwer zu spezifizieren sind. Selbst für die Validierung eines spezifischen Teilsystems erfordert die Identifikation eines Orakels erheblichen Aufwand und setzt domänenspezifisches Wissen voraus. Bei selbstlernenden Systemen mit individueller Benutzerinteraktion wird diese Aufgabe noch anspruchsvoller, da das Verhalten der Benutzer stark variieren kann. Aufgrund dieser Variabilität ist es nicht möglich, sämtliche potenziellen Benutzerinteraktionen mit herkömmlichen Testmethoden und festgelegten Testfallspezifikationen abzudecken.

Eine Möglichkeit, das Testorakel Problem zu vermeiden, stellt die Verwendung des Metamorphen Testens dar. Metamorphe Tests betrachten das Problem des Testorakels aus einer Perspektive, die von anderen Teststrategien nicht verwendet wird. Diese Herangehensweise unterscheidet sich von traditionellen Teststrategien, indem sie nicht jeden Testfall isoliert betrachtet, sondern die Ergebnisse mehrerer Testfälle und deren Beziehungen zueinander analysiert.

In dieser Dissertation wird die Methode des Metamorphen Testens in Kombination mit szenariobasierten Ansätzen für selbstlernende Systeme mit Benutzeraktionen untersucht. Zur systematischen Untersuchung dieser Methodik und zur Erreichung der Ziele der Dissertation wurden folgende Forschungsfragen formuliert:

- Wie können wiederverwendbare Testfälle für personalisierte Systeme mit Benutzeraktionen spezifiziert werden?
- Wie können synthetisch realistische Testdaten mit personenspezifischen Benutzeraktionen erzeugt werden?
- Wie kann die Ambiguität im individuellen Benutzerverhalten selbstlernender Systeme durch ein Testorakel abgebildet werden?

Die Forschungsfragen dieser Dissertation wurden durch eine methodische Herangehensweise adressiert, welche die Definition benutzerspezifischer Szenarien (siehe Kapitel 6), die synthetische Generierung von Testdaten (siehe Kapitel 7) und die Entwicklung spezifischer metamorpher Relationen für Benutzeraktionen umfasst (siehe Kapitel 9).

In Kapitel 6 wurde eine Methode zur Definition von Szenarien mit Benutzeraktionen entwickelt. Hierbei wurde das etablierte PEGASUS 6-Ebenen-Modell adaptiert und um zwei zusätzliche Ebenen erweitert: die Ebene der Streckeninformationen und die Ebene der Benutzeraktionen. Basierend auf dieser erweiterten Szenariodefinition wurden Merkmale und Ausprägungen in einer morphologischen Matrix festgelegt. Dabei wurde zwischen Merkmalen für den Kontext und Merkmalen für die Benutzeraktionen unterschieden. Aus der morphologischen Matrix wurden sechs Kombinationen von Ausprägungen für verschiedene Wegzwecke definiert.

Das in Kapitel 7 definierte Szenariomodell und die daraus abgeleiteten Wegzwecke bilden die Grundlage für die Szenarioerzeugung. Im Kern der Szenarienerzeugung steht das Tool CAGEN, das sowohl den Kontext als auch die Benutzeraktionen nach einem Providerprinzip generiert. Dieses Prinzip ermöglicht sowohl das modulare Erzeugen von Instanzen zur Bereitstellung von Testdaten als auch die sequenzielle Erstellung dieser Daten, wodurch Abhängigkeiten zwischen den Providern modelliert werden können.

Zur Erzeugung realistischer synthetischer Daten wurden die Kontextdaten in drei Klassen unterteilt: Fahrzeugdaten, Umweltdaten und Benutzerdaten. Diese Klassifizierung erlaubt es, die Auswirkungen der Umweltdaten auf das Fahrzeug oder den Benutzer abzubilden. Basierend auf den erzeugten Kontextdaten wurden realistische Benutzeraktionen generiert, die spezifisch auf den jeweiligen Kontext abgestimmt sind. Mithilfe von CAGEN konnten synthetische Testdaten für die zuvor durch die morphologische Matrix ausgewählten Wegzwecke erzeugt werden.

Kapitel 9 dieser Dissertation konzentriert sich auf das Metamorphe Testen und insbesondere auf die Definition metamorpher Relationen. Zu diesem Zweck wurde ein dreistufiges Modell entwickelt. Die ersten abstrakte Ebene identifiziert wiederkehrende Muster, welche die Formen der metamorphen Relationen charakterisieren. Diese Muster werden mittels Clusteranalysen visualisiert. Auf der zweiten Ebene wurden gemäß dieser Muster acht MMCB entwickelt, die spezifizieren, welche Eigenschaften und Aspekte das zu testende System aufweisen muss. Die dritte Ebene beinhaltet die Durchführung des metamorphen Tests, wobei die definierten Szenarien dem SuT präsentiert und anhand der MMCBs validiert wurde.

Als SuT wurde zum einen ein Gaussian Mixture Model der Firma Mercedes Benz (AC-Modell) herangezogen, das als Black-Box-System vorliegt. Zusätzlich wurden zwei Versionen eines KNNs verwendet, welche durch einen sogenannten Shadow Mode erweitert wurden (KNN+), um eine parallele Validierung und Leistungsüberwachung ohne Beeinträchtigung der eigentlichen Betriebsprozesse zu ermöglichen. Diese drei ML-Modelle wurden anhand der definierten acht MMCBs anhand drei Szenarien („Arbeit“, „Erledigung“ und „Begleitung“) validiert.

Die Forschungsergebnisse zeigen, dass das AC-Modell in den drei evaluierten Szenarien lediglich eine von acht MMCBs erfüllt. Im Vergleich dazu zeigt das KNN-Modell eine verbesserte Performance, indem es sieben der acht MMCBs einhält. Das optimierte KNN+-Modell stellt das effektivste maschinelle Lernmodell dar, da es alle acht MMCBs in allen getesteten Szenarien gewährleistet.

Abschließend wurde das Ziel der Validierung selbstlernender Systeme in Verbindung mit Benutzerinteraktionen erfolgreich erreicht. Dies wurde durch die Entwicklung eines Konzepts zur Integration von Szenarien mittels metamor-

phem Testen und durch die Evaluierung anhand drei maschineller Lernmodelle demonstriert.

11.2 Wissenschaftlicher Beitrag

Der wissenschaftliche Beitrag dieser Dissertation ist in den folgenden Punkten zusammengefasst:

- **Erweiterung der Szenariodefinition für selbstlernende Systeme mit Benutzerinteraktion:** Die Dissertation erweitert bestehende Ansätze zur Szenariodefinition, indem sie spezifisch auf die Interaktion zwischen Benutzern und selbstlernenden Systemen eingeht. Dies ermöglicht die Erzeugung von benutzerbezogenen Testszenarien und eine Beurteilung der Systemleistung.
- **CAGEN-Tool zur modularen Erzeugung von synthetischen Testdaten:** Entwicklung eines Tools zur modularen und flexiblen Erzeugung von synthetischen Testdaten, das die Erstellung datenintensiver Testszenarien für maschinelle Lernmodelle unterstützt.
- **Erstellung eines Ebenenmodells für clusterbasierte metamorphe Beziehungen:** Entwicklung eines dreistufigen Ebenenmodells, das abstrakte Musteridentifikation, die Definition spezifischer Verhaltenscluster und die praktische Testdurchführung abbildet.
- **Definition und Evaluation von acht MMCBs anhand von drei ML-Modellen:** Demonstration der Anwendung der entwickelten Methoden durch die Evaluierung von MMCBs anhand von maschinellen Lernmodellen, unterstützt durch eine Clustervisualisierung zur Validierung.

11.3 Ausblick

Die in Abschnitt 8.2 beschriebenen, prototypisch entwickelten Provider von CAGEN sind speziell auf die Anforderungen der Automobilbranche zugeschnitten. In dieser Domäne sind insbesondere Informationen zur Position, Zeit und Geschwindigkeit von Bedeutung. Allerdings gibt es auch andere relevante Kontextinformationen, die für eine SSIB notwendig sein können. Ein

Beispiel stellt die Validierung von Funktionen wie dem automatisierten Öffnen der Fenster, bei der die Anzahl der um das Egofahrzeug befindlichen Fahrzeuge eine Rolle spielt.

Für zukünftige Dissertationen könnte die Erweiterung der Provider auf andere Domänen von Nutzen sein. Beispielsweise könnten Large Language Models (LLMs) zur Generierung und Validierung natürlicher Sprachinteraktionen eingesetzt werden. Diese Erweiterung würde erlauben, Metamorphes Testen auch in textbasierten Anwendungen anzuwenden. In solchen Szenarien wären spezielle Provider erforderlich, die nicht physikalische Signale, sondern Wörter und sprachliche Konstruktionen bereitstellen.

Die Anwendung des metamorphen Testens in Kombination mit spezifischen Providern bietet für Convolutional Neural Networks (CNNs) und Bilddaten Potenzial. CNNs sind in der Bildverarbeitung verbreitet und werden unter anderem in der medizinischen Bildanalyse eingesetzt. In der medizinischen Bildanalyse ist die Genauigkeit und Zuverlässigkeit der CNNs von Bedeutung. MT kann eingesetzt werden, um sicherzustellen, dass die Modelle zuverlässige Diagnosen liefern, auch wenn die Bilddaten variieren. Dies könnte durch die Entwicklung von Providern geschehen, die medizinische Bilddaten unter verschiedenen Bedingungen und mit unterschiedlichen Anomalien generieren.

In Kapitel 9 wurde ein dreistufiges konzeptuelles Modell zur Entwicklung und Implementierung von metamorphen Testprozessen vorgestellt (siehe Abbildung 9.1). Der Übergang zwischen den Abstraktionsebenen erfolgt derzeit durch Expertenwissen. Ein Prozess, der beschreibt, wie die Abstraktionsebene ohne Expertenwissen durchlaufen werden können, würde die Erstellung von metamorphen Beziehungen für weitere Domänen öffnen. Dies könnte die Anwendung des Metamorphen Testens auf eine Vielzahl selbstlernender Systeme erweitern und dadurch die Validierung dieser Systeme signifikant optimieren.

Abkürzungsverzeichnis

ART Adaptives Random Testing 73, *Glossar:* ART

ATO Abgeleitete Testorakel 51, 52, 56, 57, 61, 62, 74, 75, *Glossar:* ATO

BDD Binary Decision Diagrams 63, *Glossar:* BDD

CAGEN Context and Action Generation 111, 112

CNN Convolutional Neural Network 17, *Glossar:* CNN

DL Deep Learning 15–17, 28, 70, *Glossar:* DL

ITO Implizite Testorakel 52, 53, 56, *Glossar:* ITO

JSON JavaScript Object Notation 113, *Glossar:* JSON

k-NN k-Nearest Neighbor 59, 65, 70, *Glossar:* k-NN

KI Künstliche Intelligenz 1–5, 15, 16, *Glossar:* KI

KNN Künstliche Neuronale Netze 17, 22, 24–27, 32, 65, 66, 72, 140–142, 185, *Glossar:* KNN

MaaS Mobilität als Dienstleistung 1, *Glossar:* MaaS

MB Metamorphen Beziehungen 66–73, 79, 81, 82, 84, 85, 88, 143–145, 148, 149, 158, 162, 163, 165, 225, *Glossar:* MB

MBUX Mercedes-Benz User Experience 2, *Glossar:* MBUX

MiD Mobilität in Deutschland 95, 98

ML Machinelles Lernen 2, 15, 17, 65, *Glossar:* ML

MLP Multilayer Perceptron 22, 23, 25, 26, 28, 29, *Glossar:* MLP

MMCB Muster Metamorpher Cluster Beziehungen 71, 145, 146, 148, 149, 151, 153, 164, 172, 173, 178, 179, 185, *Glossar:* MMCB

$\mathbf{MMCB_{ID}}$ MMCB die eine Identität darstellt 147

$\mathbf{MMCB_{LR}}$ MMCB die eine leere Menge darstellt 147

$\mathbf{MMCB_{TL}}$ MMCB die eine Teilmenge darstellt 147

MT Metamorphes Testen 66–69, 73, 78, 79, 81, 82, 84, 85, 87, 88, 143, 179, 187, 225, *Glossar:* MT

NB Naive-Bayes-Algorithmus 59, 65, 70, *Glossar:* NB

NVP N-Versionen-Programmierung 52, 53, *Glossar:* NVP

OTA Over-The-Air 51, *Glossar:* OTA

PCA Hauptkomponentenanalyse 145, 146, *Glossar:* PCA

ReLU Rectified Linear Unit 141, *Glossar:* ReLU

SAE Society of Automotive Engineers 1, 4, *Glossar:* SAE

SSIB Selbstlernende Systeme mit individuellen Benutzeraktionen 7–9, 56, 71, 75–78, 81–84, 88–91, 99, 111, 138, 145, 149, 151, 154–156, 164, 170, 175, 177, 183, 186

STO spezifiziertes Testorakel 48, 50, 52, 56, 59, 63, *Glossar:* STO

SuT System under Test 47, 49, 50, 52, 54, 59, 68, 74, 77, 82–84, 87, 88, 98, 99, 101, 106, 129, 138, 148, 149, 153, 158, 159, 161, 165, 172, 178, 179, 185, *Glossar:* SuT

SVM Support Vector Machine 65, *Glossar:* SVM

UML Unified Modeling Language 49, *Glossar:* UML

VDM-SL Vienna Development Method - Specification Language 49, *Glossar:* VDM-SL

Glossar

Alloy Alloy ist eine deklarative Spezifikationssprache, die für die Modellierung, Spezifikation und Analyse von Software- und Informationssystemstrukturen entwickelt wurde. Charakteristisch für Alloy ist ihre Nutzung einer relationalen Logik, die eine Darstellung von Strukturen und Beziehungen zwischen diesen ermöglicht. Die Sprache unterstützt das Design und die Überprüfung von Modellen durch den Alloy Analyzer, ein Tool, das automatische Analysen durchführt, um Konsistenz, Korrektheit und andere spezifizierte Eigenschaften der Modelle zu verifizieren. 49

ART Adaptives Random Testing ist eine verbesserte Methode des zufälligen Testens, die die Wahrscheinlichkeit der Fehlerfindung durch strategisch verteilte Testfälle erhöht. 73

Assertions Assertions sind programmatische Anweisungen, die überprüfen, ob bestimmte Bedingungen im Programmcode erfüllt sind. Wenn diese Bedingungen wahr sind, setzt das Programm seine Ausführung fort; sind sie jedoch falsch, deuten sie auf Fehler im Programm hin, was zum Abbruch der Ausführung führt. 50, 51

ATO Abgeleitete Testorakel beziehen sich auf Testorakel, die aus anderen Quellen oder Testfällen abgeleitet werden. Diese Art von Testorakel basiert auf spezifischen Regeln, Algorithmen oder Modellen, die aus bereits bekannten Daten oder Testergebnissen extrahiert werden. 51

BDD Binary Decision Diagrams, sind Datenstrukturen, die benutzt werden, um boolesche Funktionen zu repräsentieren und zu manipulieren. Sie ermöglichen eine effiziente Implementierung von Algorithmen zur Durchführung von logischen Operationen und haben sich insbesondere für die Modellierung und Verifikation von Systemen in Bereichen wie Hardware-Design und formaler Verifikationsmethodik als nützlich erwiesen. 63

CNN Ein Convolutional Neural Network ist ein Typ von tiefem neuronalem Netz, das primär für visuelle Aufgaben wie Bild- und Videoerkennung verwendet wird. Es besteht aus Faltungs-, Pooling- und vollständig verbundenen Schichten, die es ermöglichen, räumliche Hierarchien von Merkmalen effizient zu erkennen und zu lernen. 17

Contracts Im Kontext der Softwareentwicklung und -sicherheit bezieht sich der Begriff „Contracts“ auf formale, spezifizierte Vereinbarungen zwischen verschiedenen Komponenten eines Systems oder zwischen einem System und seinen Nutzern. Diese Verträge definieren Erwartungen bezüglich der Funktionsweise von Methoden, Modulen oder Klassen in Form von Vorbedingungen, Nachbedingungen und Invarianten. 51

DL Eine Untergruppe des maschinellen Lernens, die auf künstlichen neuronalen Netzen mit mehreren Schichten (tiefen Netzen) basiert. Diese Technik ermöglicht es Maschinen, Muster in Datenmengen zu erkennen und zu interpretieren, ähnlich der Art und Weise, wie Menschen lernen und Entscheidungen treffen. 15, 16

ITO Implizite Testorakel sind nicht explizit angegeben oder formalisiert, sondern basieren auf implizitem Wissen oder Verhalten. Sie können sich auf nicht direkt erkennbare Eigenschaften oder Verhaltensweisen eines Systems beziehen, die jedoch durch bestimmte Eingaben oder Situationen manifestiert werden können. 52

JSON JSON ist ein leichtgewichtiges Datenformat, das für den Datenaustausch zwischen Servern und Webanwendungen verwendet wird. JSON-Dateien speichern Daten in einem strukturierten Format, das sowohl von Menschen lesbar als auch von Maschinen effizient verarbeitbar ist. 113

k-NN Der k-Nearest Neighbor (k-NN) ist ein nicht-parametrisches, überwachtes Lernverfahren, das sowohl für Klassifizierungs- als auch für Regressionsaufgaben eingesetzt wird. Es klassifiziert Objekte basierend auf den k nächstgelegenen Beispielen im Merkmalsraum, wobei ’k’ eine positive ganze Zahl ist. Die Zuordnung zu einer Klasse erfolgt durch

eine Mehrheitsabstimmung der ’k’ nächsten Nachbarn, wobei die Distanz zwischen den Punkten üblicherweise mittels Euklidischer Distanz berechnet wird. 59

KI Künstliche Intelligenz (KI) bezeichnet den Bereich der Informatik, der sich mit der Schaffung von Maschinen oder Programmen beschäftigt, die Fähigkeiten aufweisen, die traditionell menschliche Intelligenz erfordern. Dazu gehören das Verstehen natürlicher Sprache, das Erkennen von Bildern und Mustern. 1

KNN Ein Modell für maschinelles Lernen, inspiriert von der Struktur und Funktionsweise des menschlichen Gehirns, das aus miteinander verbundenen Knoten oder „Neuronen’"besteht. Diese Netze können lernen, Beziehungen in Daten zu erkennen, indem sie Eingabedaten durch mehrere Schichten von Neuronen verarbeiten, wobei jedes Neuron einen Teil der Datenverarbeitung übernimmt. 17

Kreuzreferenzierung In Bezug auf Testorakel bezeichnet Kreuzreferenzierung die Methode, bei der Ergebnisse oder Verhaltensweisen eines zu testenden Systems (System under Test, SuT) mit voneinander unabhängigen Quellen oder Testfällen verglichen werden. Ziel ist es, die Konsistenz und Korrektheit der Testergebnisse zu überprüfen, indem sie mit alternativen Implementierungen, spezifizierten Anforderungen oder vorherigen Versionen des Systems kreuzreferenziert werden. 57

MaaS Mobilität als Dienstleistung ist ein Konzept, bei dem verschiedene Verkehrsmittel und Mobilitätsdienste in einer einzigen, benutzerfreundlichen Plattform integriert werden. Nutzer können über eine App oder eine Website auf verschiedene Transportmöglichkeiten wie öffentliche Verkehrsmittel, Carsharing, Fahrradleihsysteme und Ridesharing-Dienste zugreifen, planen und bezahlen. 1

MB Metamorphe Beziehungen sind ein Konzept im Bereich des Softwaretestens, das zur Überwindung der Herausforderungen bei der Verifikation von Systemen ohne festgelegte Testorakel eingesetzt wird. Sie basieren auf der Identifizierung von Eigenschaften oder Mustern, die zwischen verschiedenen Eingaben und ihren zugehörigen Ausgaben eines Programms konsistent bleiben sollten. 66

MBUX Das Infotainmentsystem Mercedes-Benz User Experience (MBUX) ist eine fortschrittliche Benutzerschnittstelle, die in Mercedes-Benz-Fahrzeugen zum Einsatz kommt. Es kombiniert künstliche Intelligenz mit einem intuitiven Bedienkonzept, um ein personalisiertes Fahrerlebnis zu bieten. 2

Mealy-Maschine Eine Mealy-Maschine ist ein endlicher Automat, bei dem die Ausgabe von dem aktuellen Zustand und dem aktuellen Eingabewert abhängt. Das bedeutet, dass die Ausgabe direkt auf eine Eingabe reagiert und sich mit jeder Eingabeänderung ändern kann, selbst wenn der Automat im gleichen Zustand bleibt. 49, 50

ML Ein Teilbereich der künstlichen Intelligenz, der Algorithmen verwendet, um Computern die Fähigkeit zu geben, aus Daten zu lernen und Vorhersagen oder Entscheidungen ohne explizite Programmierung zu treffen. 2

MLP Ein Multilayer Perceptron ist ein Feedforward-Künstliches Neuronales Netz, das aus mehreren Schichten besteht, darunter Eingabe-, verborgene und Ausgabeschichten, mit vollständig verbundenen Neuronen zwischen den Schichten. Es nutzt Aktivierungsfunktionen, um Nichtlinearität einzuführen, und wird für Aufgaben wie Klassifikation und Regression eingesetzt. 22

MMCB Muster metamorpher Cluster Beziehungen beziehen sich auf spezifische Testmuster innerhalb der metamorphen Testmethodik, die verwendet werden, um die Korrektheit eines selbstlernenden Systems zu überprüfen. MMCBs fokussieren sich auf die Beziehungen zwischen verschiedenen Clustern von Testfällen. 71

Moore-Maschine Eine Moore-Maschine-Maschine ist ein endlicher Automat, bei die Ausgabe ausschließlich vom aktuellen Zustand des Automaten abhängt und nicht von der Eingabe, die zu diesem Zustand geführt hat. Das bedeutet, dass die Ausgabe nur ändert, wenn der Automat in einen anderen Zustand wechselt. 49, 50

MT Metamorphes Testen verwendet metamorphe Beziehungen zur Verifizierung von Software, insbesondere wenn klassische Testorakel fehlen oder schwer zu definieren sind. Es eignet sich vor allem für Systeme mit indeterministischem Verhalten, wie sie im maschinellen Lernen auftre-

ten. Diese Methode analysiert systematische und vorhersehbare Verhaltensänderungen der Software in Reaktion auf gezielte Änderungen der Eingaben, um die Funktionalität zu bewerten. 66

NB Der Naive-Bayes-Algorithmus ist ein probabilistischer Klassifizierungsansatz, der auf dem Bayes-Theorem basiert und die Annahme trifft, dass die Merkmale eines Datensatzes unabhängig voneinander sind, gegeben die Klassenzugehörigkeit. Diese Vereinfachung ermöglicht die Modellierung von Klassifizierungsproblemen, insbesondere bei textuellen Daten wie der E-Mail-Spam-Erkennung und der Sentiment-Analyse. 59

NVP Die N-Versionen-Programmierung ist eine Technik zur Fehlertoleranz in der Softwareentwicklung, bei der N verschiedene Implementierungen desselben Algorithmus erstellt und gleichzeitig ausgeführt werden. Die Ergebnisse der einzelnen Implementierungen werden dann mit einem Abstimmungsverfahren verglichen, um das korrekte Ergebnis zu ermitteln. 52

operationelle Semantik Die operationelle Semantik beschreibt die Bedeutung eines Programms durch die Angabe seiner Auswirkungen auf den Zustand einer abstrakten Maschine. Die abstrakte Maschine repräsentiert den Computer, auf dem das Programm ausgeführt wird. Der Zustand der abstrakten Maschine umfasst den Speicher, die Register und den Programmzähler. 49

OTA Over-the-Air bezeichnet die drahtlose Übertragung von Softwareaktualisierungen, Konfigurationen oder Daten an Endgeräte wie Smartphones, IoT-Geräte oder eingebettete Systeme. Diese Technologie ermöglicht die Fernwartung und -aktualisierung von Gerätesoftware, einschließlich Betriebssystemen, Sicherheitspatches und Anwendungssoftware, ohne physischen Zugriff auf das Gerät. 51

PCA Hauptkomponentenanalyse (Principal Component Analysis, PCA) ist ein statistisches Verfahren zur Reduktion der Dimensionalität von Datensätzen, während die Varianz und die wichtigsten Muster der Daten erhalten bleiben. PCA transformiert die ursprünglichen Variablen in eine neue

Menge von unkorrelierten Variablen, die als Hauptkomponenten bezeichnet werden. Diese Hauptkomponenten werden so gewählt, dass die erste Hauptkomponente die größtmögliche Varianz der Daten erklärt, die zweite Hauptkomponente die größtmögliche verbleibende Varianz erklärt 145

ReLU Die Rectified Linear Unit (ReLU) ist eine Aktivierungsfunktion, die häufig in neuronalen Netzwerken verwendet wird. Sie definiert die Ausgabe eines Neurons als den maximalen Wert zwischen null und dem Eingabewert. Mathematisch wird die ReLU-Funktion durch die Formel $ReLU(x) = max(0, x)$ beschrieben. Dies bedeutet, dass alle negativen Eingabewerte auf null gesetzt werden, während positive Eingabewerte unverändert bleiben. 141

SAE Die SAE-Level (Society of Automotive Engineers) beschreiben die verschiedenen Stufen der Automatisierung bei Fahrzeugen, die von Stufe 0 (keine Automatisierung) bis Stufe 5 (vollständige Automatisierung) reichen. Stufe 0 bedeutet, dass das Fahrzeug keinerlei Unterstützungssysteme für den Fahrer hat. Stufe 5 steht für vollständige Automatisierung unter allen Bedingungen, bei der keine menschliche Eingriffe erforderlich sind und das Fahrzeug ohne Lenkrad oder Bedienelemente auskommen kann. 1

STO Ein spezifiziertes Testorakel ist ein Testwerkzeug in der Softwareentwicklung, das auf einer exakten Vorgabe von Ausgabeerwartungen basiert, um die Übereinstimmung der tatsächlichen Ergebnisse eines Systems mit seinen definierten Anforderungen und Spezifikationen zu überprüfen. 48

SuT Das System Under Test, auf Deutsch auch als zu testendes System bezeichnet, ist das System, welches im Rahmen eines Softwaretests auf seine korrekte Funktionsweise überprüft wird. 47

SVM Ein überwachtes maschinelles Lernmodell, das für Klassifikations- und Regressionsaufgaben verwendet wird. Es funktioniert, indem es die Datenpunkte in einen höherdimensionalen Raum projiziert und eine optimale Trennlinie (oder Hyperebene) findet, die die verschiedenen Klassen mit einem maximalen Abstand voneinander trennt. 65

symbolischer Vergleich Symbolischer Vergleich ist eine Technik, bei der die Ausführungen zweier Versionen einer Software – einer fehlerbehafteten und einer aktualisierten – nicht durch direkte Auswertung mit spezifischen Werten, sondern mittels symbolischer Ausdrücke gegenübergestellt werden. Diese Methode erlaubt die Analyse des Softwareverhaltens über ein Spektrum potenzieller Eingaben hinweg, ohne jede Eingabemöglichkeit explizit ausführen zu müssen. 62

UML Unified Modeling Language ist eine standardisierte Modellierungssprache in der Softwaretechnik für Spezifikation, Visualisierung, Konstruktion und Dokumentation von Softwaresystemartefakten und Geschäftsprozessen. UML bietet grafische Notationstechniken zur Darstellung von Systemstrukturen (Klassendiagramme, Objektdiagramme, Paketdiagramme), Verhaltensweisen (Anwendungsfalldiagramme, Aktivitätsdiagramme, Zustandsdiagramme) und Interaktionen zwischen Systemkomponenten (Sequenzdiagramme, Kommunikationsdiagramme). 49

VDM-SL Die Vienna Development Method - Specification Language (VDM-SL) ist eine formale Spezifikationssprache, die für die Beschreibung und Analyse von Softwaresystemen konzipiert wurde. Sie nutzt mathematische Konstrukte, insbesondere Mengenlehre und Logik, um sowohl das Verhalten als auch die Datenstrukturen von Systemen exakt zu modellieren. VDM-SL ermöglicht die Erstellung abstrakter und konkreter Modelle, die zur Spezifikation von Systemanforderungen, zur Verifikation der Systemkorrektheit und zur Unterstützung der inkrementellen Systementwicklung verwendet werden können. 49

Z-Notation Die Z-Notation, auch als „Z“ bezeichnet, ist eine formale Notationssprache, die für die Spezifikation und Modellierung von Softwaresystemen entwickelt wurde. Ihr Fokus liegt auf der eindeutigen Festlegung von Anforderungen und Verhalten eines Systems auf formaler Ebene, bevor mit der eigentlichen Programmierung begonnen wird. 49

12 Anhang

12.1 Morphologische Matrix aller Wegzwecke und daraus resultierende Szenarien

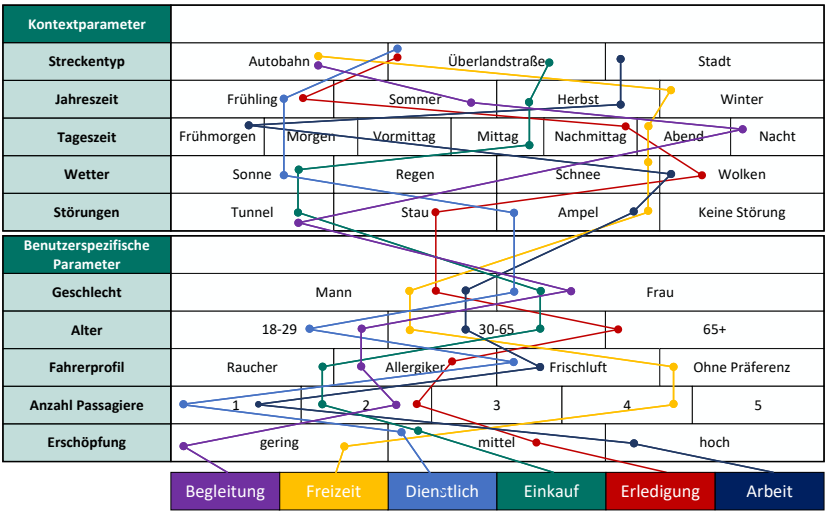

Abbildung 12.1: Darstellung der Szenarien anhand der morphologischen Matrix

12.2 Evaluation der MMCBs für das Szenario „Erledigung“

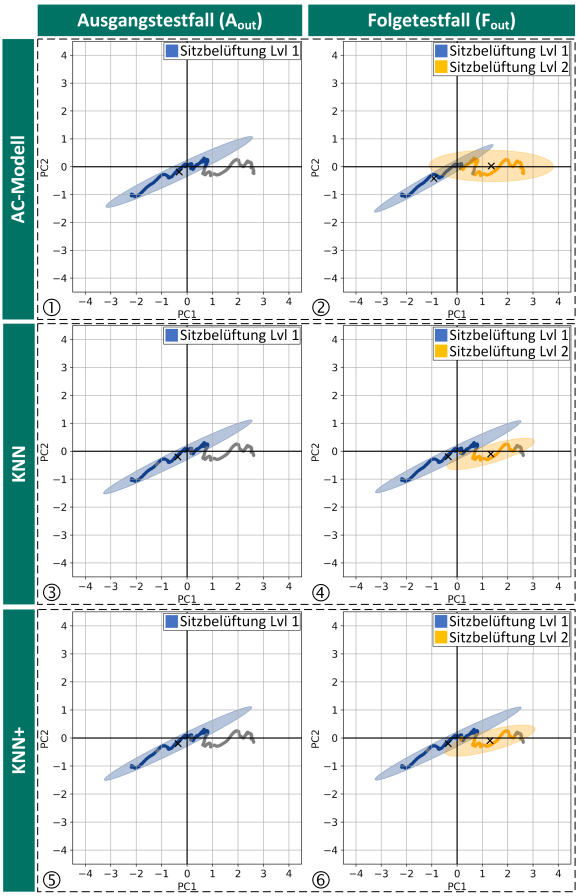

Abbildung 12.2: Ergebnisse Szenario Erledigung - $MMCB_{TL}$ 1.1

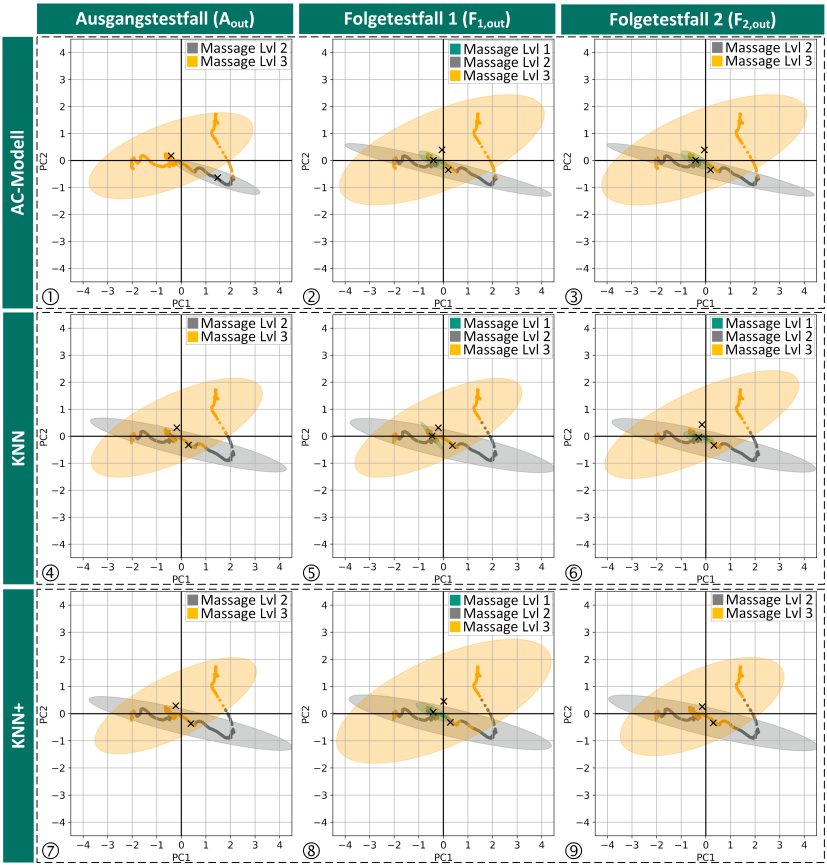

Abbildung 12.3: Ergebnisse Szenario Erledigung - $MMCB_{ID}1.2$

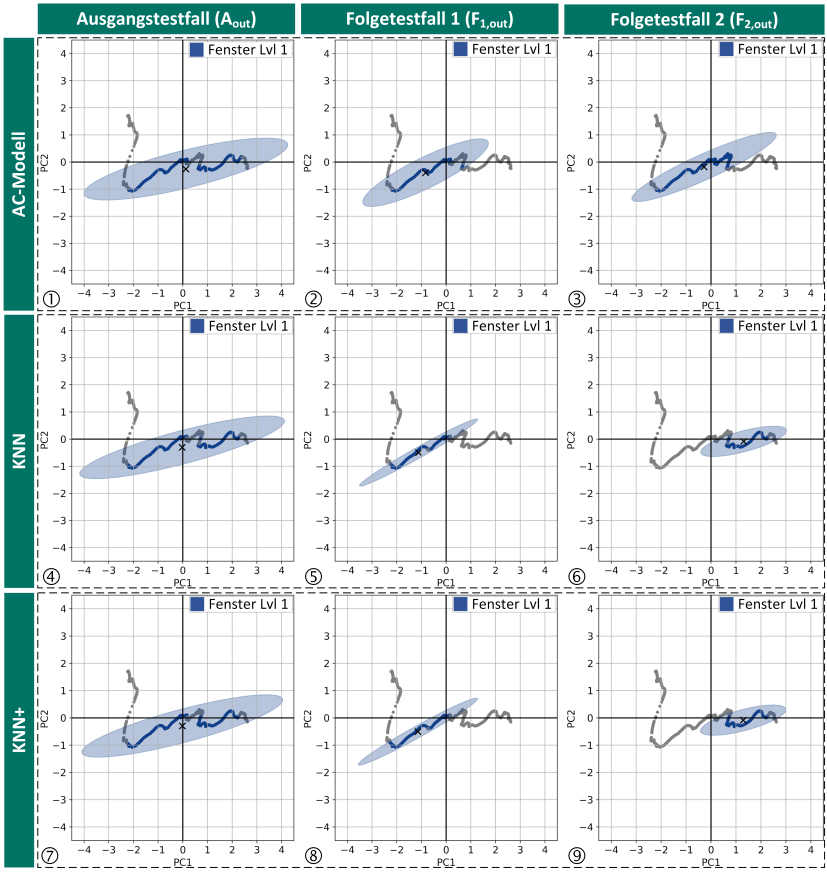

Abbildung 12.4: Ergebnisse Szenario Erledigung - $MMCB_{TL}$ 1.3

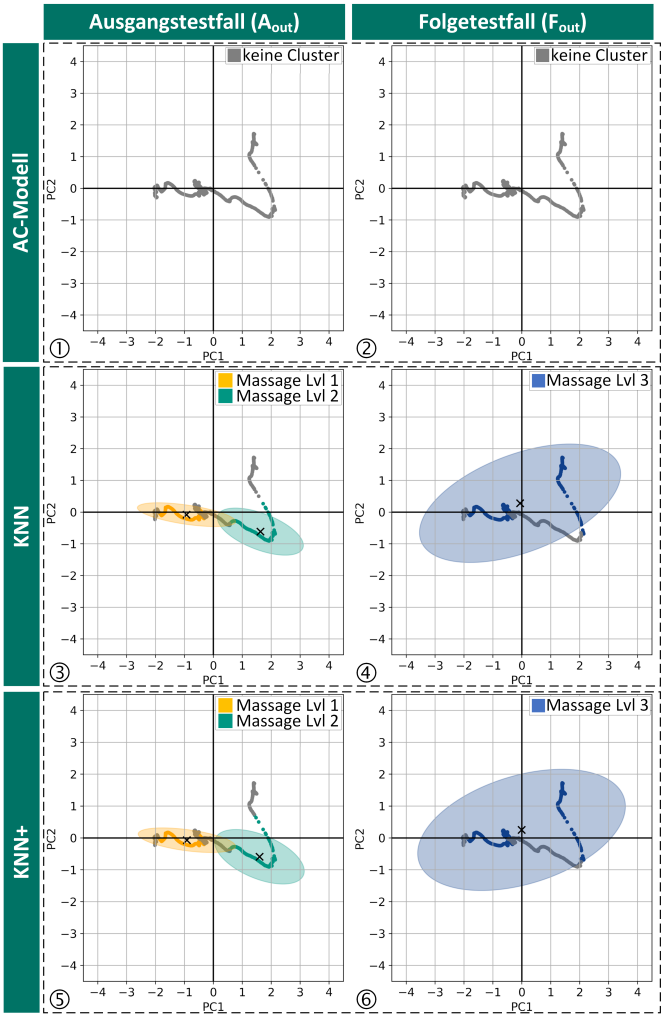

Abbildung 12.5: Ergebnisse Szenario Erledigung - $MMCB_{LR}$ 2.1

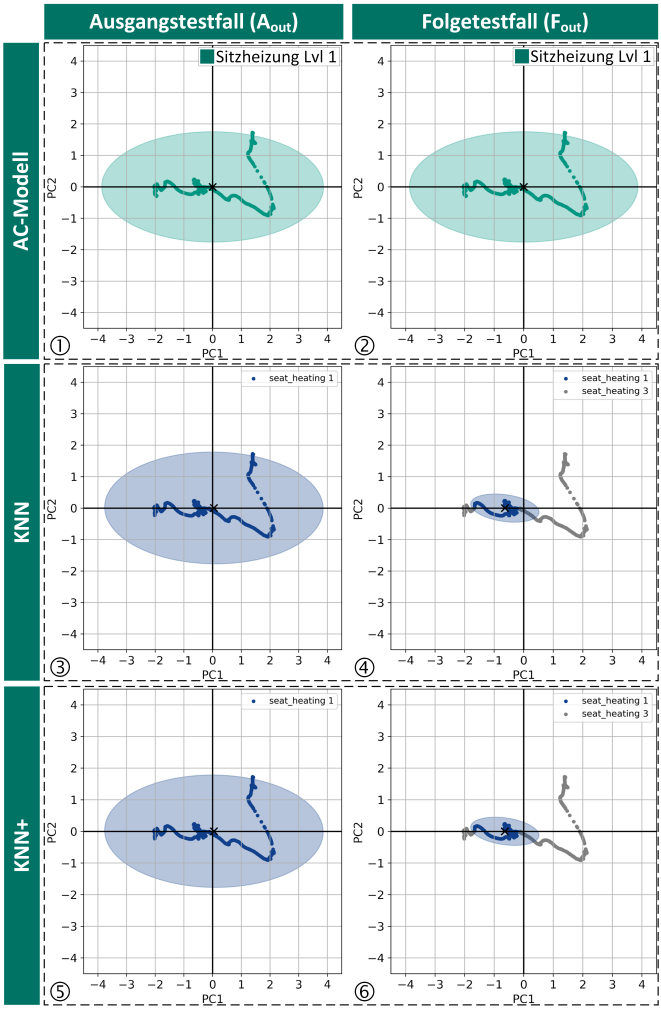

Abbildung 12.6: Ergebnisse Szenario Erledigung - $MMCB_{TL}$ 2.2

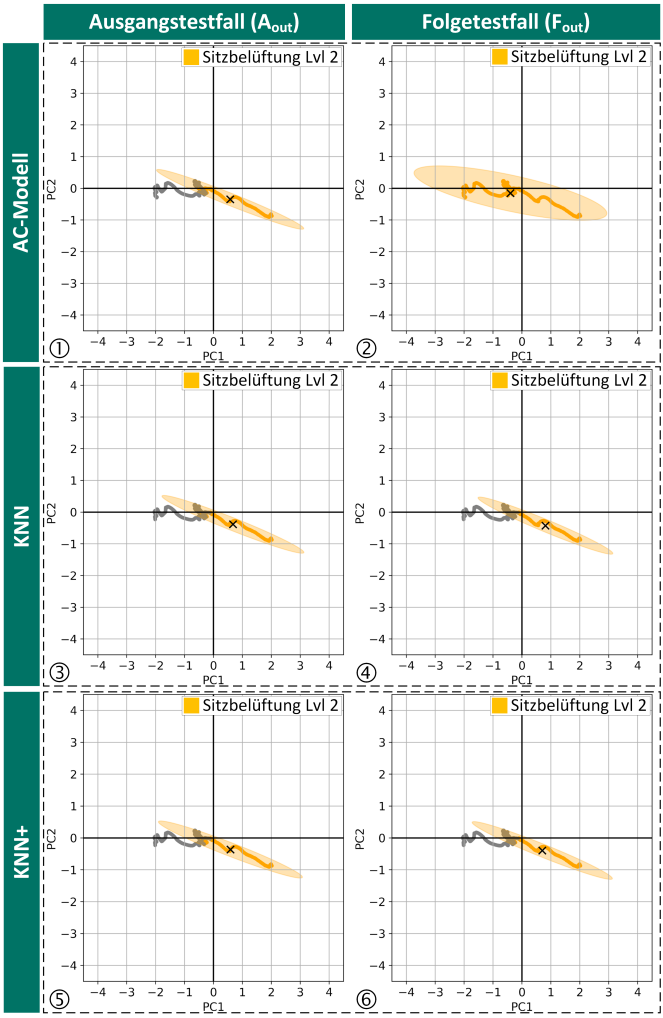

Abbildung 12.7: Ergebnisse Szenario Erledigung - $MMCB_{ID}2.3$

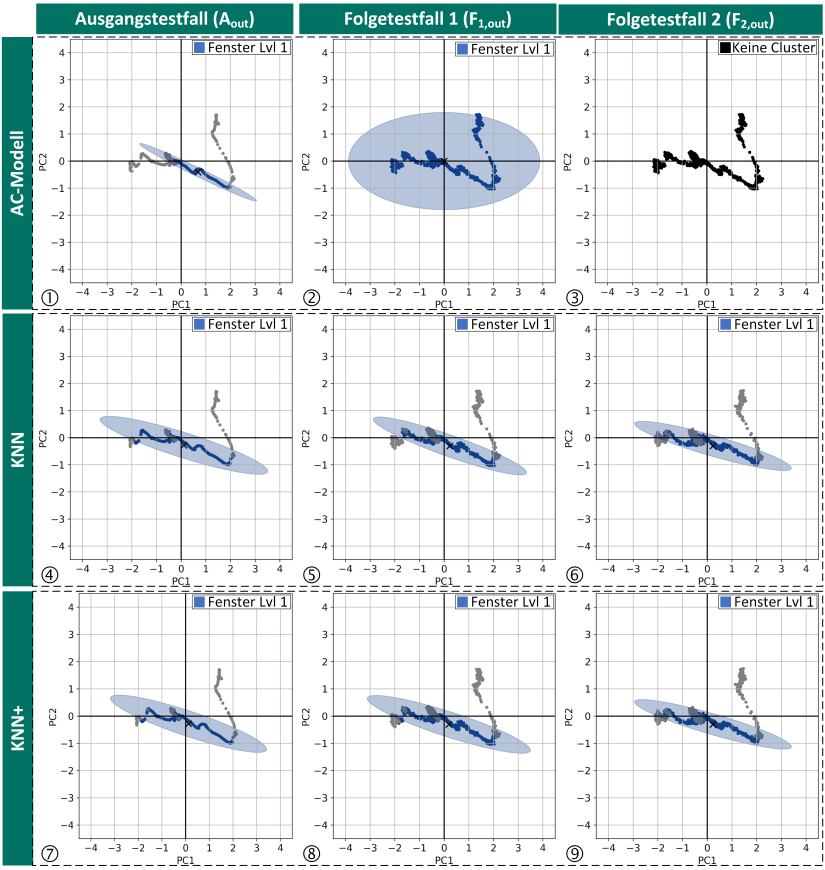

Abbildung 12.8: Ergebnisse Szenario Erledigung - $MMCB_{ID}3.1$

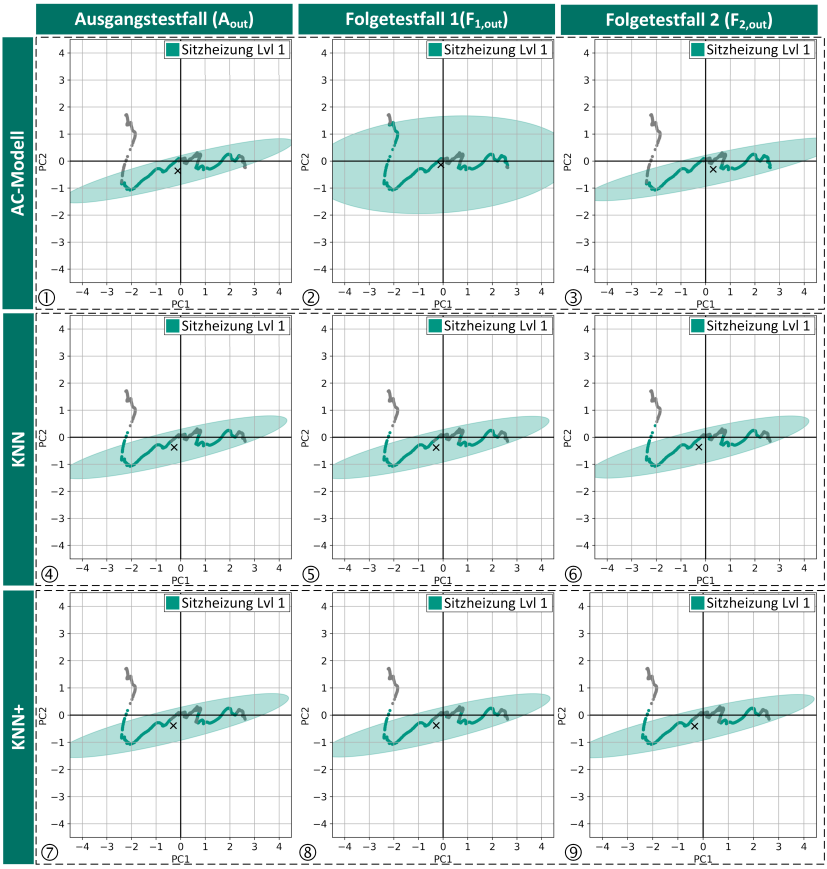

Abbildung 12.9: Ergebnisse Szenario Erledigung - $MMCB_{ID}3.2$

12.2.1 Tabelle der Genauigkeiten für Szenario „Erledigung“

Tabelle 12.1: Tabelle der Genauigkeiten für Ausgangstestfall und Folgetestfälle im Vergleich zu den Ergebnissen der MMCBs (nach Tabelle 10.1) für Szenario „Erledigung“

	ML-Modell	**Ausgangstestfall (A_{out})**	**Folgetestfall ($F_{1,out}$)**	**Folgetestfall ($F_{2,out}$)**	**Szenario „Arbeit“ (nach Tabelle 10.1)**
$MMCB_{TL}$1.1	AC	0,8023	0,8033	0,5126	✓
	KNN	0,9556	0,9654	0,9772	✓
	KNN+	0,9167	0,9599	0,9686	✓
$MMCB_{ID}$1.2	AC	0,7716	0,8298	0,8298	✗
	KNN	0,9383	0,7427	0,8421	✗
	KNN+	0,9556	0,9113	0,9286	✓
$MMCB_{TL}$1.3	AC	1	0,982	0,3128	✗
	KNN	0,9837	0,9881	0,9794	✓
	KNN+	0,9783	0,9816	0,9881	✓
$MMCB_{LR}$2.1	AC	0,5517	0,5517	-	✗
	KNN	0,9902	0,9827	-	✓
	KNN+	0,9859	0,9935	-	✓
$MMCB_{TL}$2.2	AC	0,4619	0,3657	-	✗
	KNN	0,4508	0,9491	-	✓
	KNN+	0,4508	0,9475	-	✓
$MMCB_{TL}$2.3	AC	0,8033	0,3393	0,5571	✗
	KNN	0,9556	0,9405	0,9783	✓
	KNN+	0,9394	0,9794	0,9751	✓
$MMCB_{ID}$3.1	AC	0,5708	0,5137	0,0011	✗
	KNN	0,9805	0,9697	0,8064	✓
	KNN+	0,9697	0,9545	0,8735	✓
$MMCB_{ID}$3.2	AC	0,9957	0,8107	0,8964	✗
	KNN	0,9729	0,9762	0,9589	✓
	KNN+	0,9567	0,9471	0,9437	✓

12.3 Evaluation der MMCBs für das Szenario „Begleitung“

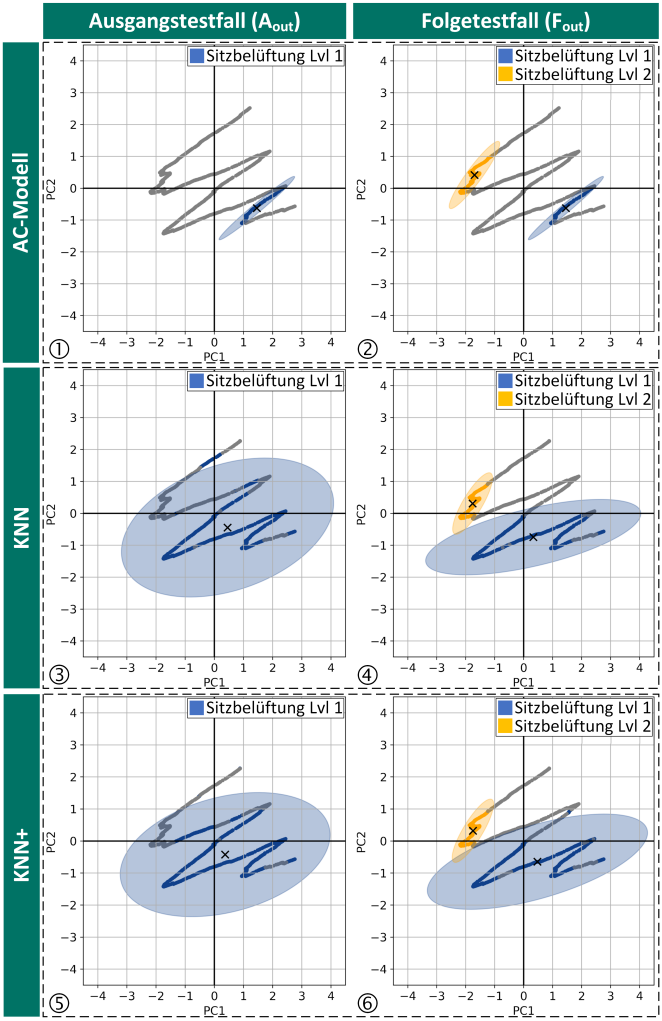

Abbildung 12.10: Ergebnisse Szenario Begleitung - $MMCB_{TL}$ 1.1

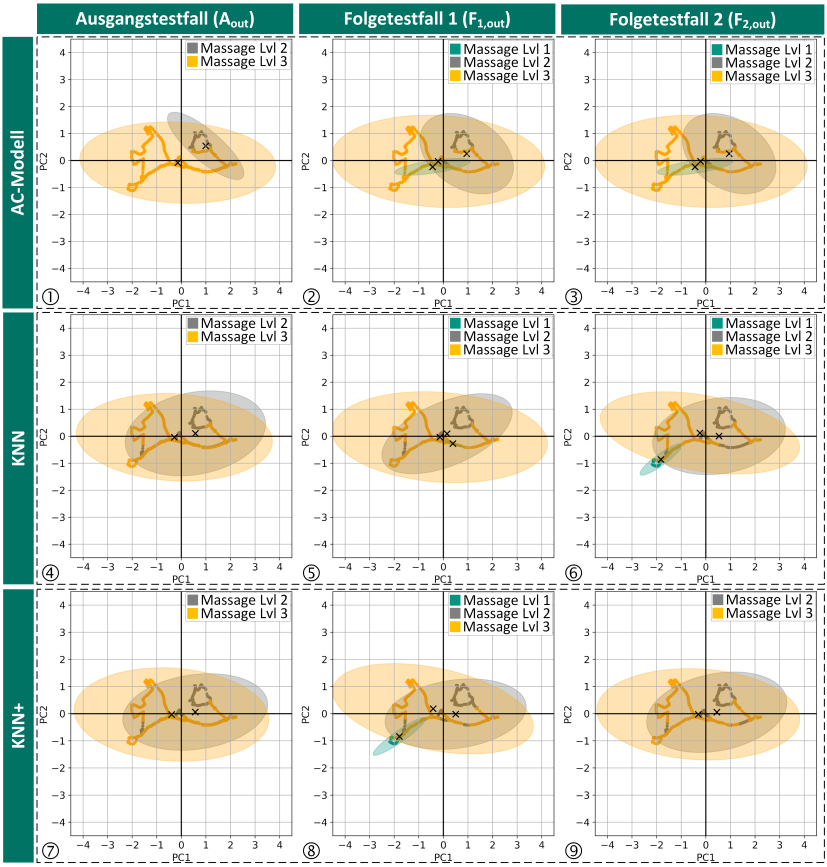

Abbildung 12.11: Ergebnisse Szenario Begleitung - $MMCB_{ID}1.2$

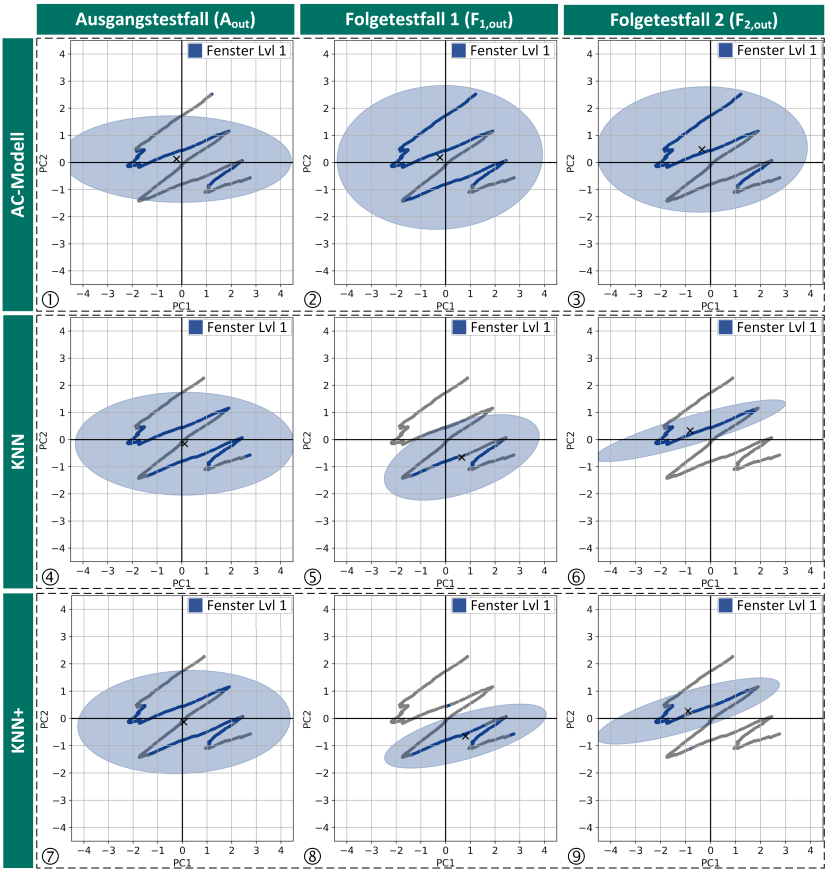

Abbildung 12.12: Ergebnisse Szenario Begleitung - $MMCB_{TL}$ 1.3

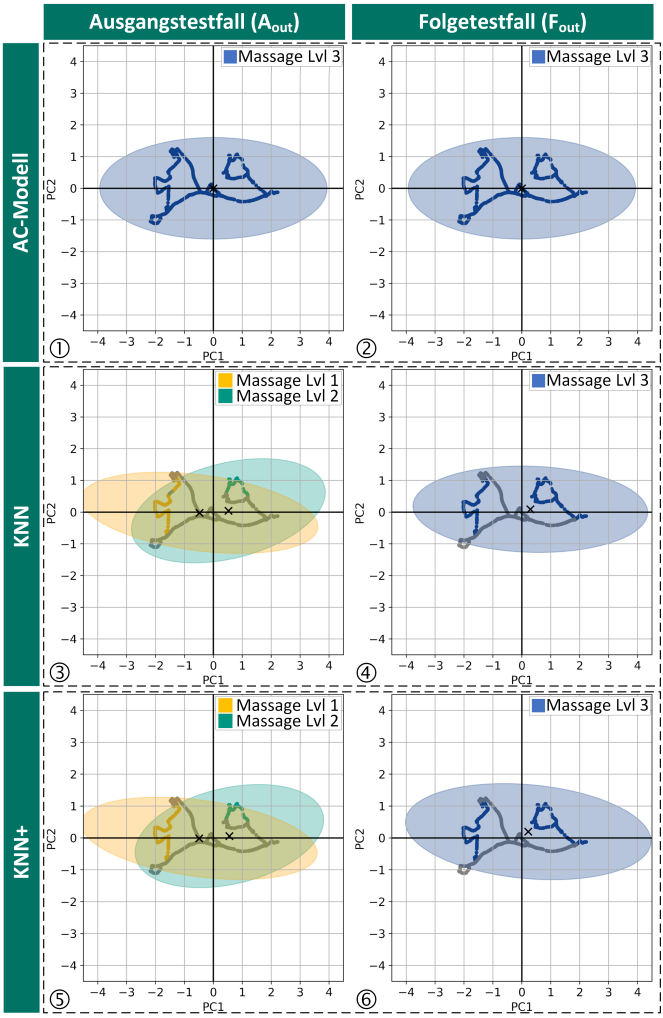

Abbildung 12.13: Ergebnisse Szenario Begleitung - $MMCB_{LR}2.1$

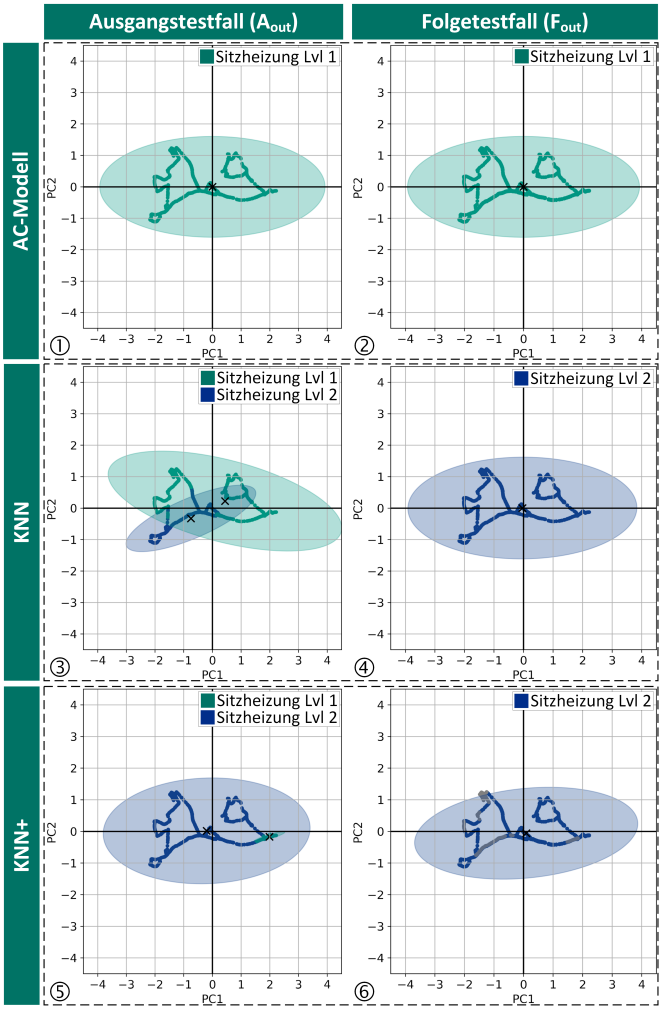

Abbildung 12.14: Ergebnisse Szenario Begleitung - $MMCB_{TL}2.2$

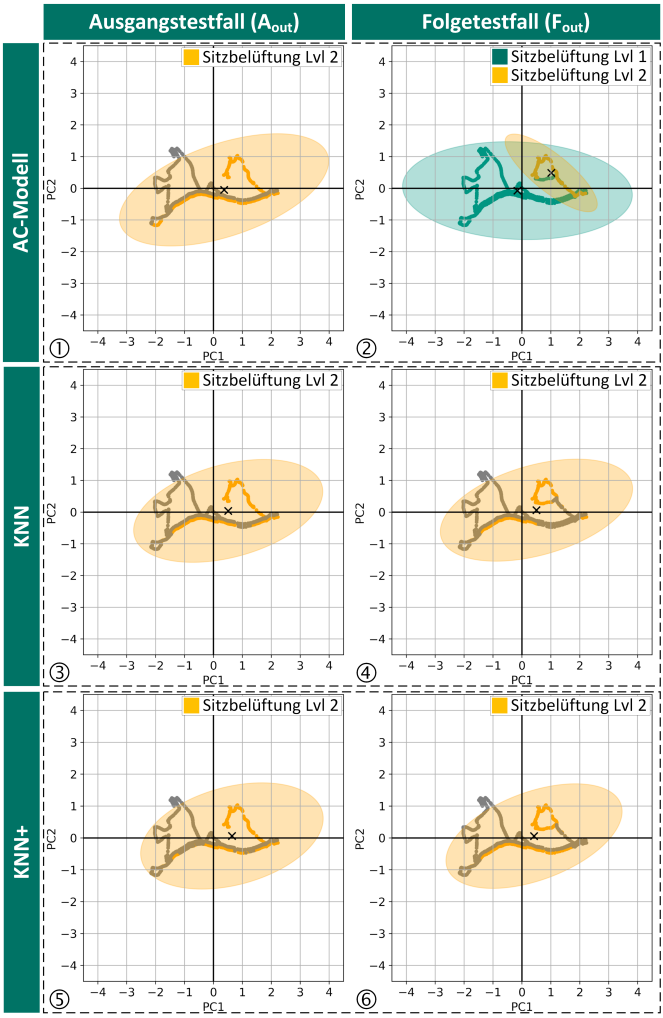

Abbildung 12.15: Ergebnisse Szenario Erledigung - $MMCB_{ID}2.3$

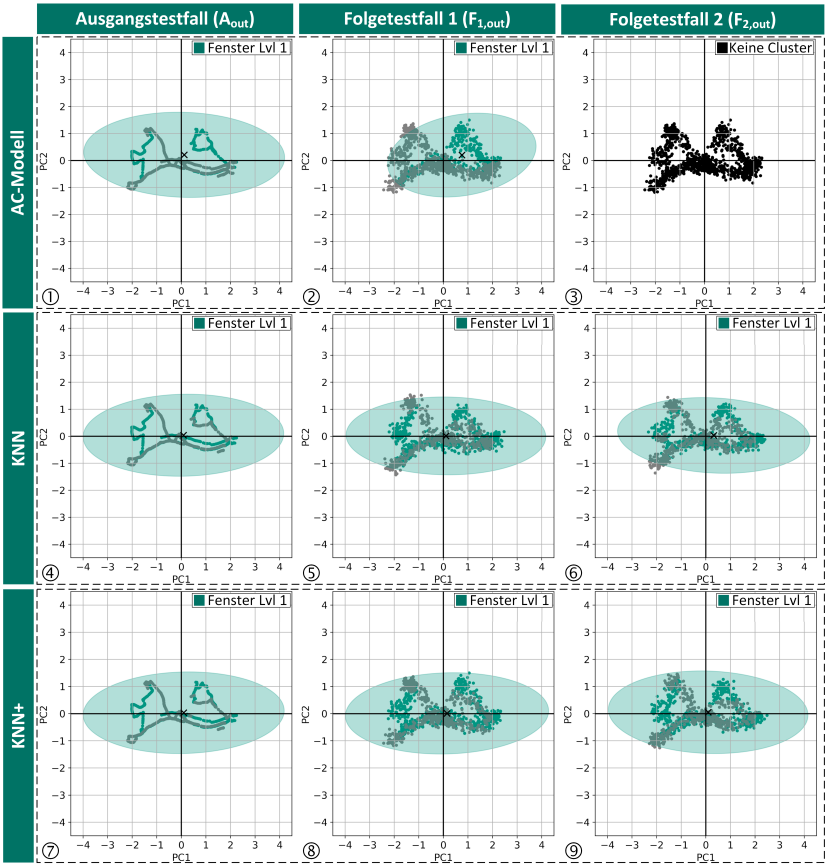

Abbildung 12.16: Ergebnisse Szenario Erledigung - $MMCB_{ID}3.1$

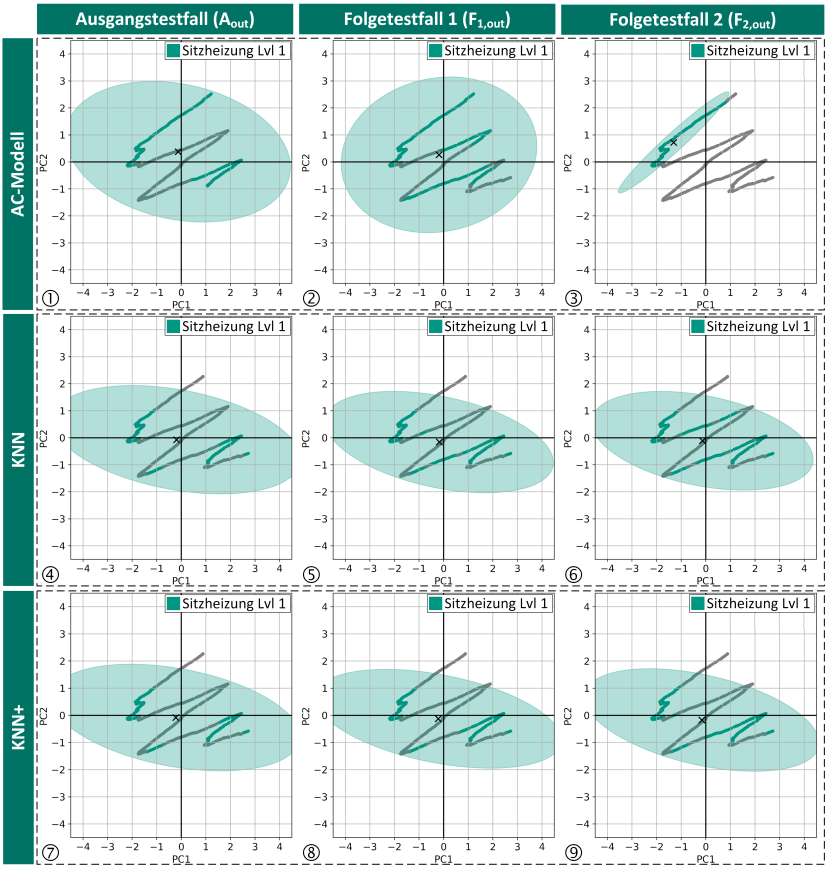

Abbildung 12.17: Ergebnisse Szenario Erledigung - $MMCB_{ID}3.2$

12.3.1 Tabellen der Genauigkeiten für Szenario „Begleitung“

Tabelle 12.2: Tabelle der Genauigkeiten für Ausgangstestfall und Folgetestfälle im Vergleich zu den Ergebnissen der MMCBs (nach Tabelle 10.1) für Szenario „Begleitung“

	ML-Modell	**Ausgangstestfall** (A_{out})	**Folgetestfall** ($F_{1,out}$)	**Folgetestfall** ($F_{2,out}$)	**Szenario „Arbeit“ (nach Tabelle 10.1)**
MMCB$_{TL}$1.1	AC	0,9262	0,6471	0,6471	✓
	KNN	0,9666	0,9085	0,8999	✓
	KNN+	0,9628	0,8942	0,9133	✓
MMCB$_{ID}$1.2	AC	0,7451	0,7572	0,7572	✗
	KNN	0,7447	0,6561	0,7838	✗
	KNN+	0,8609	0,8104	0,7828	✓
MMCB$_{TL}$1.3	AC	0,8244	0,5536	0,7329	✗
	KNN	0,9847	0,9657	0,8009	✓
	KNN+	0,9781	0,9933	0,8523	✓
MMCB$_{LR}$2.1	AC	0,2745	0,2745	-	✗
	KNN	0,9752	0,9191	-	✓
	KNN+	0,9723	0,9419	-	✓
MMCB$_{TL}$2.2	AC	0,0009	0,2922	-	✗
	KNN	0,5991	0,2971	-	✓
	KNN+	0,0742	0,4495	-	✓
MMCB$_{TL}$2.3	AC	0,9262	0,1316	0,3333	✗
	KNN	0,9828	0,9561	0,8447	✓
	KNN+	0,9552	0,9685	0,8361	✓
MMCB$_{ID}$3.1	AC	0,5406	0,5901	0,0009	✗
	KNN	0,9857	0,9599	0,8723	✓
	KNN+	0,9676	0,9571	0,8809	✓
MMCB$_{ID}$3.2	AC	0,7181	0,7647	0,6694	✗
	KNN	0,8809	0,9304	0,8733	✓
	KNN+	0,8847	0,9152	0,9295	✓

12.4 Tabellen der Euklidische Distanz der Eventcluster

Tabelle 12.3: Euklidische Distanz der visualisierten Eventcluster für Szenario „Arbeit“

		Euklidische Distanz zwischen		
	ML-Modell	A_{out} **zu** $F_{1,out}$	A_{out} **zu** $F_{2,out}$	**Eventcluster für**
MMCB$_{TL}$1.1	AC	1,1516	1,1516	Sitzbelüftung Stufe 1
	KNN	1,2046	1,1293	Sitzbelüftung Stufe 1
	KNN+	1,1975	1,1998	Sitzbelüftung Stufe 1
MMCB$_{ID}$1.2	AC	0,0345	0,0311	Massage Stufe 2
		0,3966	0,0167	Massage Stufe 3
	KNN	0,1871	0,1601	Massage Stufe 2
		0,3419	0,4925	Massage Stufe 3
	KNN+	0,1967	0,1261	Massage Stufe 2
		0,6585	0,1795	Massage Stufe 3
MMCB$_{TL}$1.3	AC	0,0071	0,0071	Fenster Stufe 1
	KNN	2,2775	1,0546	Fenster Stufe 1
	KNN+	2,0391	1,3898	Fenster Stufe 1
MMCB$_{LR}$2.1	AC	-	-	Massage
	KNN	∞	-	Massage
	KNN+	∞	-	Massage
MMCB$_{TL}$2.2	AC	0,0008	-	Sitzheizung Stufe 1
	KNN	0,9312	-	Sitzheizung Stufe 1
	KNN+	1,4679	-	Sitzheizung Stufe 1
MMCB$_{TL}$2.3	AC	1,3035	-	Sitzbelüftung Stufe 1
	KNN	0,0011	-	Sitzbelüftung Stufe 1
	KNN+	0,0086	-	Sitzbelüftung Stufe 1
MMCB$_{ID}$3.1	AC	1,1865	-	Fenster Stufe 1
	KNN	0,2126	0,2066	Fenster Stufe 1
	KNN+	0,0624	0,0731	Fenster Stufe 1
MMCB$_{ID}$3.2	AC	0,2255	0,4302	Sitzheizung Stufe 1
	KNN	0,0197	0,0147	Sitzheizung Stufe 1
	KNN+	0,0202	0,0448	Sitzheizung Stufe 1

Tabelle 12.4: Euklidische Distanz der visualisierten Eventcluster für Szenario „Erledigung“

		Euklidische Distanz zwischen		
	ML-Modell	A_{out} **zu** $F_{1,out}$	A_{out} **zu** $F_{2,out}$	**Eventcluster für**
MMCB$_{TL}$1.1	AC	0,0065	0,7613	Sitzbelüftung Stufe 1
	KNN	0,0414	0,0975	Sitzbelüftung Stufe 1
	KNN+	0,1089	0,1208	Sitzbelüftung Stufe 1
MMCB$_{ID}$1.2	AC	1,3139	1,3144	Massage Stufe 2
		0,4241	0,4245	Massage Stufe 3
	KNN	0,0814	0,0397	Massage Stufe 2
		0,0194	0,1184	Massage Stufe 3
	KNN+	0,1172	0,0894	Massage Stufe 2
		0,2954	0,0829	Massage Stufe 3
MMCB$_{TL}$1.3	AC	0,9708	0,4111	Fenster Stufe 1
	KNN	1,0934	1,2665	Fenster Stufe 1
	KNN+	1,1227	1,3952	Fenster Stufe 1
MMCB$_{LR}$2.1	AC	-	-	Massage
	KNN	∞	-	Massage
	KNN+	∞	-	Massage
MMCB$_{TL}$2.2	AC	0,0009	-	Sitzheizung Stufe 1
	KNN	0,6832	-	Sitzheizung Stufe 1
	KNN+	0,6853	-	Sitzheizung Stufe 1
MMCB$_{TL}$2.3	AC	0,9951	-	Sitzbelüftung Stufe 1
	KNN	0,1441	-	Sitzbelüftung Stufe 1
	KNN+	0,1194	-	Sitzbelüftung Stufe 1
MMCB$_{ID}$3.1	AC	0,7956	-	Fenster Stufe 1
	KNN	0,2711	0,4413	Fenster Stufe 1
	KNN+	0,0731	0,1138	Fenster Stufe 1
MMCB$_{ID}$3.2	AC	0,0104	0,4599	Sitzheizung Stufe 1
	KNN	0,1164	0,0655	Sitzheizung Stufe 1
	KNN+	0,3394	0,3351	Sitzheizung Stufe 1

Tabelle 12.5: Euklidische Distanz der visualisierten Eventcluster für Szenario „Begleitung“

		Euklidische Distanz zwischen		
	ML-Modell	A_{out} **zu** $F_{1,out}$	A_{out} **zu** $F_{2,out}$	**Eventcluster für**
MMCB$_{TL}$1.1	AC	0,7206	0,7205	Sitzbelüftung Stufe 1
	KNN	0,4738	0,5666	Sitzbelüftung Stufe 1
	KNN+	0,5427	0,3841	Sitzbelüftung Stufe 1
MMCB$_{ID}$1.2	AC	0,3031	0,3021	Massage Stufe 2
		0,0839	0,0839	Massage Stufe 3
	KNN	0,1058	0,1415	Massage Stufe 2
		0,0522	0,1705	Massage Stufe 3
	KNN+	0,2921	0,3121	Massage Stufe 2
		0,4851	0,2586	Massage Stufe 3
MMCB$_{TL}$1.3	AC	0,0618	0,3778	Fenster Stufe 1
	KNN	0,9657	1,6185	Fenster Stufe 1
	KNN+	0,8993	1,7823	Fenster Stufe 1
MMCB$_{LR}$2.1	AC	-	-	Massage
	KNN	∞	-	Massage
	KNN+	∞	-	Massage
MMCB$_{TL}$2.2	AC	-	-	Sitzheizung Stufe 1
	KNN	0,7832	-	Sitzheizung Stufe 1
	KNN+	0,3008	-	Sitzheizung Stufe 1
MMCB$_{TL}$2.3	AC	0,8375	-	Sitzbelüftung Stufe 1
	KNN	0,0244	-	Sitzbelüftung Stufe 1
	KNN+	0,2204	-	Sitzbelüftung Stufe 1
MMCB$_{ID}$3.1	AC	0,8392	-	Fenster Stufe 1
	KNN	0,0407	0,1635	Fenster Stufe 1
	KNN+	0,1936	0,1982	Fenster Stufe 1
MMCB$_{ID}$3.2	AC	0,1293	1,2271	Sitzheizung Stufe 1
	KNN	0,0887	0,0697	Sitzheizung Stufe 1
	KNN+	0,0457	0,1093	Sitzheizung Stufe 1

12.5 Abdeckung der Anforderungen durch diese Dissertation

Tabelle 12.6: Anforderungen an die Szenariodefinition

ID	Beschreibung	Abdeckung	Abschnitt(e)
AS1	Aufbauend auf die Szenariodefinition nach ISO 34501 muss die Szenariodefinition für die Beschreibung eines Szenarios für selbstlernende Komfortfunktionen mit Benutzerinteraktion angepasst werden.	✓ Pegasus Adaption	6.1 (S. 91)
AS2	Die Szenariendefinition soll die Identifikation von repräsentativen Szenarien ermöglichen, die die charakteristische Nutzung eines selbstlernenden Systems durch einen Benutzer abbilden.	✓ Szenariodefinition durch Expertenwissen	6 (S. 89)
AS3	Die Szenariodefinition muss die Einbeziehung relevanter Randbedingungen und Variablen ermöglichen, welche das Verhalten der selbstlernenden System beeinflussen könnten.	✓ Merkmalsauswahl und Morphologische Matrix	6.2 (S. 93), 6.2.1 (S. 95)

Tabelle 12.7: Anforderungen an die Szenarioerzeugung

ID	Beschreibung	Abdeckung	Abschnitt(e)
AT1	Die Datenerzeugung soll einerseits den untersuchten Kontext darstellen und andererseits Benutzerinteraktionen abbilden.	✓ Context- Action-Generation (CAGEN)	7.1 (S. 100), 7.3 (S. 105)
AT2	Die Erzeugung von Testdaten soll mithilfe eines dezentralen Ansatzes erfolgen, um ein modulares Hinzufügen oder Entfernen von Datenquellen zu ermöglichen.	✓ Modularität	7.2 (S. 102)
AT3	Die synthetisch erzeugten Testdaten sollen der empirischen Datenverteilung entsprechen, die in der realen Welt beobachtet wird.	✓ Ideal- /Sensor- /Benutzer Kontext	7.1 (S. 100)
AT4	Die Szenarioerzeugung soll die Generierung von Testdaten ermöglichen, die ein realitätsnahes Verhalten eines Benutzers abbilden.	✓ Benutzerinteraktionen	7.3 (S. 105)

Tabelle 12.8: Konzeptionelle Anforderungen an das Metamorphe Testen

ID	Beschreibung	Abdeckung	Abschnitt(e)
AT1	Die Verwendung des MT muss das Testorakel-Problem verhindern, das insbesondere beim Testen selbstlernender Systeme mit Benutzeraktionen auftritt.	✓ Diskussion des Standes der Technik	4.4 (S. 74)
AT2	Es müssen systematisch MB identifiziert werden, die sich auf die Benutzeraktionen und das Verhalten der Funktion auswirken. Diese Beziehungen sollten geeignet sein, um potenzielle Veränderungen im Benutzerverhalten aufzudecken.	✓ Abstraktionsebenenmodell	9.1 (S. 144)
AT3	Die Resultate des MT sollen so gestaltet sein, dass sie eine automatisierte Analyse und Interpretation ermöglichen.	✓ Muster Metamorpher Cluster Beziehungen	9.1.1 (S. 145)

Literaturverzeichnis

[1] Bormann, René and Fink, Philipp and Holzapfel, Helmut and Rammler, Stephan and Sauter-Servaes, Thomas and Tiemann, Heinrich and Waschke, Thomas and Weirauch, Boris. Die Zukunft der deutschen Automobilindustrie. *Transformation by Disaster oder by Design*, 3, 2018.

[2] Shantanu Ingle and Madhuri Phute. Tesla autopilot: semi autonomous driving, an uptick for future autonomy. *International Research Journal of Engineering and Technology*, 3(9):369–372, 2016.

[3] Tobias Gödecke, Christoph Schildhauer, Frank Weinert, and Arthur Frick. The plug-in hybrid powertrain for compact cars from mercedes-benz. *MTZ worldwide*, 80(11):32–39, 2019.

[4] Tim Dutton. An overview of national ai strategies, 2018.

[5] Stanford Graduate School of Business. Msx future forum: Exploring trends that are changing the future, 2020-08-31T06:35:24.000Z.

[6] Xavier Ferràs-Hernández. The future of management in a world of electronic brains. *Journal of Management Inquiry*, 27(2):260–263, 2018.

[7] Roger C. Mayer, James H. Davis, and F. David Schoorman. An integrative model of organizational trust. *The Academy of Management Review*, 20(3):709, 1995.

[8] Weiquan Wang, Lingyun Qiu, Dongmin Kim, and Izak Benbasat. Effects of rational and social appeals of online recommendation agents on cognition- and affect-based trust. *Decision Support Systems*, 86:48–60, 2016.

[9] Andrew Ng. The batch: Google ai explains itself, neural net fights bias, ai demoralizes champions, solar power heats up, 2019.

[10] Peter Gola. Datenschutz-grundverordnung. *Aufl., München*, 2018.

[11] ISO/IEC/IEEE 24765:2010. ISO/IEC/IEEE International Standard - Systems and software engineering – Vocabulary. *ISO/IEC/IEEE 24765:2010(E)*, pages 1–418, 2010.

[12] IEEE Computer Society. Ieee standard for system, software, and hardware verification and validation, 2016.

[13] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229, 1959.

[14] Stefan Schmerler. *Softwaretest in der Praxis: Grundlagen, Methoden und Technologien*. epubli, 2020.

[15] Earl T. Barr, Mark Harman, Phil McMinn, Muzammil Shahbaz, and Shin Yoo. The oracle problem in software testing: A survey. *IEEE Transactions on Software Engineering*, 41(5):507–525, 2014.

[16] Julian Moeser. Starke KI, schwache KI - Was kann künstliche Intelligenz?, 2018.

[17] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT Press, Cambridge, Massachusetts and London, England, 2016.

[18] Fraunhofer-Allianz Big Data. Zukunftsmarkt künstliche intelligenz–potenziale und anwendungen, 2018.

[19] Vladimir Cherkassky and Filip M Mulier. *Learning from data: concepts, theory, and methods*. John Wiley & Sons, 2007.

[20] Christian Gärtner. Smart hrm: Digitale tools für die personalarbeit. In *Smart HRM*, pages 1–4. Springer, 2020.

[21] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[22] Marco Iansiti and Karim R Lakhani. *Competing in the age of AI: strategy and leadership when algorithms and networks run the world*. Harvard Business Press, 2020.

[23] Marina Sokolova and Guy Lapalme. A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, 45(4):427–437, 2009.

[24] R.A. Fisher. UCI machine learning repository, iris data set, 1936.

[25] Tao Li, Chengliang Zhang, and Shenghuo Zhu. Empirical studies on multi-label classification. In *8th [i.e. 18th] IEEE International Conference on Tools with Artificial Intelligence, 2006*, pages 86–92, Los Alamitos, Calif., 2006. IEEE Computer Society.

[26] Stefan Richter. *Statistisches und maschinelles Lernen: Gängige Verfahren im Überblick*. Lehrbuch. Springer Berlin Heidelberg, Berlin, Heidelberg, 2019.

[27] Marc Weber. Untersuchungen zur anomalieerkennung in automotive steuergeräten durch verteilte observer mit fokus auf die plausibilisierung von kommunikationssignalen.

[28] Stuart J. Russell and Peter Norvig. *Artificial intelligence: A modern approach*. Prentice Hall series in artificial intelligence. Prentice Hall, Upper Saddle River, NJ, 2. ed., internat. ed. edition, 2003.

[29] Aurélien Géron. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow*. Ö'Reilly Media, Inc.", 2022.

[30] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386–408, 1958.

[31] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[32] Karl Steinbuch. Die lernmatrix. *Kybernetik*, 1(1):36–45, 1961.

[33] Frank Rosenblatt. Principles of neurodynamics. perceptrons and the theory of brain mechanisms. Technical report, Cornell Aeronautical Lab Inc Buffalo NY, 1961.

[34] Stuart J. Russell and Peter Norvig. *Artificial intelligence: A modern approach*. Always learning. Pearson, Boston and Columbus and Indianapolis, third edition, global edition edition, 2016.

[35] RA Fisher. A mathematical examination of the methods of determining the accuracy of observation by the mean error, and by the mean square error. *Monthly Notices of the Royal Astronomical Society*, 80(8):758–770, 1920.

[36] Avrim Blum and Ronald L Rivest. Training a 3-node neural network is np-complete. In *Proceedings of the 1st International Conference on Neural Information Processing Systems*, pages 494–501, 1988.

[37] Alexei Botchkarev. Performance metrics (error measures) in machine learning regression, forecasting and prognostics: Properties and typology. *Interdisciplinary Journal of Information, Knowledge, and Management*, 2018.

[38] Charles E Metz. Basic principles of roc analysis. In *Seminars in nuclear medicine*, volume 8, pages 283–298. Elsevier, 1978.

[39] Frederick Winslow Taylor. *Die Grundsätze wissenschaftlicher Betriebsführung*. Oldenbourg, 1919.

[40] ISO/IEC/IEEE 12207:2017. Systems and software engineering – Software life cycle processes. 2017.

[41] Winston W. Royce. Managing the development of large software systems: concepts and techniques. In *Proceedings of the 9th international conference on Software Engineering*, pages 328–338, 1987.

[42] Barry W Boehm. *Software engineering economics*. Springer, 2002.

[43] Jean Paul Calvez, Alan Wyche, and Charles Edmundson. *Embedded real-time systems*. Wiley New York, 1993.

[44] Olegas Vasilecas and Aidas Smaizys. Business rule model integration into the model of transformation driven software development. In Janis Grundspenkis, editor, *Advances in Databases and Information Systems*, volume 5968 of *LNCS sublibrary. SL 3, Information systems and applications, incl. Internet/Web, and HCI*, pages 153–160. Springer, Berlin, 2010.

[45] Wolfgang Dröschel and Manuela Wiemers. *Das V-Modell 97: der Standard für die Entwicklung von IT-Systemen mit Anleitung für den Praxiseinsatz*. Walter de Gruyter GmbH & Co KG, 2015.

[46] Herbert Palm, Jörg Holzmann, Robert Klein, Stefan-Alexander Schneider, and Dieter Gerling. A novel approach on virtual systems prototyping based on a validated, hierarchical, modular library. *Embedded World*, 2013.

[47] Jörg Schäuffele and Thomas Zurawka. *Automotive Software Engineering: Grundlagen, Prozesse, Methoden und Werkzeuge*. Springer-Verlag, 2013.

[48] ISO/IEC. Iso/iec 25051:2014; systems and software engineering – systems and software quality requirements and evaluation (square) - requirements for quality of ready to use software product (rusp) and instructions for testing, 2014.

[49] Danhua Shao, Sarfraz Khurshid, and Dewayne E. Perry. A case for white-box testing using declarative specifications poster abstract. In *TAIC PART 2007*, page 137, Los Alamitos, CA and [Piscataway, N.J.], 2007. IEEE Computer Society and [IEEE].

[50] Stuart Reid. The art of software testing, second edition. glenford j. myers. revised and updated by tom badgett and todd m. thomas, with corey sandler. john wiley and sons, new jersey, u.s.a., 2004. isbn: 0-471-46912-2 , pp 234. *Software Testing, Verification and Reliability*, 15(2):136–137, 2005.

[51] Ankush Dadwal, Hironori Washizaki, Yoshiaki Fukazawa, Takahiro Iida, Masashi Mizoguchi, and Kentaro Yoshimura. Prioritization in automotive software testing: Systematic literature review. In *QuASoQ APSEC*, pages 52–58, 2018.

[52] DIN EN ISO. Din en iso 9000:2015 - qualitätsmanagementsysteme - grundlagen und begriffe, 2015.

[53] Srinivas Nidhra and Jagruthi Dondeti. Black box and white box testing techniques-a literature review. *International Journal of Embedded Systems and Applications (IJESA)*, 2(2):29–50, 2012.

[54] ISO/SAE DPAS 22736. Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles. 2021.

[55] Gerrit Bagschik, Till Menzel, and Markus Maurer. Ontology based Scene Creation for the Development of Automated Vehicles. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1813–1820, 2018.

[56] ISO 26262:2018. Road vehicles – functional safety, 2018.

[57] Simon Ulbrich, Till Menzel, Andreas Reschka, Fabian Schuldt, and Markus Maurer. Defining and Substantiating the Terms Scene, Situation, and Scenario for Automated Driving. In *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, pages 982–988, 2015.

[58] Gerrit Bagschik, Till Menzel, Andreas Reschka, and Markus Maurer. Szenarien für Entwicklung, Absicherung und Test von automatisierten fahrzeugen. In *11. Workshop Fahrerassistenzsysteme. Hrsg. von UniDAS e. V*, pages 125–135, 2017.

[59] Raphael Pfeffer. *Szenariobasierte simulationsgestützte funktionale Absicherung hochautomatisierter Fahrfunktionen durch Nutzung von Realdaten*. PhD thesis, Karlsruher Institut für Technologie (KIT), 2020.

[60] Schlussbericht für das Gesamtprojekt PEGASUS, 08/11/2023. https://www.pegasusprojekt.de/.

[61] Fritz Zwicky and A. G. Wilson. *New Methods of Thought and Procedure: Contributions to the Symposium on Methodologies*. Springer Berlin Heidelberg, Berlin, Heidelberg, 1967.

[62] Hilko Donker and Ludmilla Zaczek. Computerunterstützung der kreativitätstechnik "morphologische matrix". In *Workshop Gemeinschaften in Neuen Medien (GeNeMe) 2006*, pages 271–282. 2006.

[63] Ole-Johan Dahl, Edsger Wybe Dijkstra, and Charles Antony Richard Hoare. *Structured programming*. Academic Press Ltd., 1972.

[64] Earl T. Barr, Mark Harman, Phil McMinn, Muzammil Shahbaz, and Shin Yoo. The Oracle Problem in Software Testing: A Survey. *IEEE Transactions on Software Engineering*, 41(5):507–525, May 2015.

[65] Mark Utting, Alexander Pretschner, and Bruno Legeard. A taxonomy of model-based testing approaches. *Software testing, verification and reliability*, 22(5):297–312, 2012.

[66] Axel van Lamsweerde. Formal specification: a roadmap. In *Proceedings of the Conference on the Future of Software Engineering*, pages 147–159, 2000.

[67] J Michael Spivey and Jean-Raymond Abrial. *The Z notation*, volume 29. Prentice Hall Hemel Hempstead, 1992.

[68] Graeme Smith. *The Object-Z specification language*, volume 1. Springer Science & Business Media, 2012.

[69] John Fitzgerald and Peter Gorm Larsen. *Modelling systems: practical tools and techniques in software development*. Cambridge University Press, 2009.

[70] Daniel Jackson. *Software Abstractions: logic, language, and analysis*. MIT press, 2012.

[71] Fabrice Bouquet, Christophe Grandpierre, Bruno Legeard, Fabien Peureux, Nicolas Vacelet, and Mark Utting. A subset of precise uml for model-based testing. In *Proceedings of the 3rd international workshop on Advances in model-based testing*, pages 95–104, 2007.

[72] David Lee and Mihalis Yannakakis. Principles and methods of testing finite state machines-a survey. *Proceedings of the IEEE*, 84(8):1090–1123, 1996.

[73] Ashish Tiwari. Formal semantics and analysis methods for simulink stateflow models. Technical report, Citeseer, 2002.

[74] Alan Turing. Checking a large routine. In *The early British computer conferences*, pages 70–72, 1989.

[75] Alberto Sangiovanni-Vincentelli, Werner Damm, and Roberto Passerone. Taming dr. frankenstein: Contract-based design for cyber-physical systems. *European journal of control*, 18(3):217–238, 2012.

[76] Houssem Guissouma, Moritz Zink, and Eric Sax. Continuous safety assessment of updated supervised learning models in shadow mode. In *2023 IEEE 20th International Conference on Software Architecture Companion (ICSA-C)*, pages 301–308. IEEE, 2023.

[77] E. J. Weyuker. On testing non-testable programs. *The Computer Journal*, 25(4):465–470, 1982.

[78] Algirdas Avizienis. The methodology of n-version programming. *Software fault tolerance*, 3:23–46, 1995.

[79] Michael R Lyu and Y-T He. Improving the n-version programming process through the evolution of a design paradigm. *IEEE Transactions on Reliability*, 42(2):179–189, 1993.

[80] Seyed Reza Shahamiri, Wan Mohd Nasir Wan Kadir, and Siti Zaiton Mohd-Hashim. A comparative study on automated software test oracle methods. In *2009 fourth international conference on software engineering advances*, pages 140–145. IEEE, 2009.

[81] Barton P. Miller, Lars Fredriksen, and Bryan So. An empirical study of the reliability of unix utilities. *Communications of the ACM*, 33(12):32–44, 1990.

[82] Sofia Bekrar, Chaouki Bekrar, Roland Groz, and Laurent Mounier. Finding software vulnerabilities by smart fuzzing. In *2011 Fourth IEEE International Conference on Software Testing, Verification and Validation*, pages 427–430. IEEE, 2011.

[83] William M McKeeman. Differential testing for software. *Digital Technical Journal*, 10(1):100–107, 1998.

[84] Vu Le, Mehrdad Afshari, and Zhendong Su. Compiler validation via equivalence modulo inputs. *ACM Sigplan Notices*, 49(6):216–226, 2014.

[85] Mahdi Nejadgholi and Jinqiu Yang. A study of oracle approximations in testing deep learning libraries. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 785–796. IEEE, 2019.

[86] Siwakorn Srisakaokul, Zhengkai Wu, Angello Astorga, Oreoluwa Alebiosu, and Tao Xie. Multiple-implementation testing of supervised

learning software. In *Workshops at the thirty-second AAAI conference on artificial intelligence*, 2018.

[87] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. Deepxplore. In *Proceedings of the 26th Symposium on Operating Systems Principles*, ACM Digital Library, pages 1–18, New York, NY, 2017. ACM.

[88] Hung Viet Pham, Thibaud Lutellier, Weizhen Qi, and Lin Tan. Cradle: cross-backend validation to detect and localize bugs in deep learning libraries. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 1027–1038. IEEE, 2019.

[89] Zan Wang, Ming Yan, Junjie Chen, Shuang Liu, and Dongdi Zhang. Deep learning library testing via effective model generation. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 788–799, 2020.

[90] Shin Yoo and Mark Harman. Regression testing minimization, selection and prioritization: a survey. *Software testing, verification and reliability*, 22(2):67–120, 2012.

[91] Zhongxian Gu, Earl T Barr, David J Hamilton, and Zhendong Su. Has the bug really been fixed? In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*, pages 55–64, 2010.

[92] Tao Xie. Augmenting automatically generated unit-test suites with regression oracle checking. In *European Conference on Object-Oriented Programming*, pages 380–403. Springer, 2006.

[93] Mark Gabel and Zhendong Su. Symbolic mining of temporal specifications. In *Proceedings of the 30th international conference on Software engineering*, pages 51–60, 2008.

[94] Joxan Jaffar, Jorge A Navas, and Andrew E Santosa. Unbounded symbolic execution for program verification. In *Runtime Verification: Second International Conference, RV 2011, San Francisco, CA, USA, September 27-30, 2011, Revised Selected Papers 2*, pages 396–411. Springer, 2012.

[95] Caroline Lemieux, Dennis Park, and Ivan Beschastnikh. General ltl specification mining (t). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 81–92. IEEE, 2015.

[96] Jenny Abrahamson, Ivan Beschastnikh, Yuriy Brun, and Michael D Ernst. Shedding light on distributed system executions. In *Companion Proceedings of the 36th International Conference on Software Engineering*, pages 598–599, 2014.

[97] Jinlin Yang, David Evans, Deepali Bhardwaj, Thirumalesh Bhat, and Manuvir Das. Perracotta: mining temporal api rules from imperfect traces. In *Proceedings of the 28th international conference on Software engineering*, pages 282–291, 2006.

[98] Michael D Ernst, Jeff H Perkins, Philip J Guo, Stephen McCamant, Carlos Pacheco, Matthew S Tschantz, and Chen Xiao. The daikon system for dynamic detection of likely invariants. *Science of computer programming*, 69(1-3):35–45, 2007.

[99] Christoph Csallner, Nikolai Tillmann, and Yannis Smaragdakis. Dysy: Dynamic symbolic execution for invariant inference. In *Proceedings of the 30th international conference on Software engineering*, pages 281–290, 2008.

[100] Maryam Raiyat Aliabadi, Hassan Haghighi, Mojtaba Vahidi Asl, and Ramak Ghavamizadeh Meybodi. Challenges of specification mining-based test oracle for cyber-physical systems. In *2020 11th International Conference on Information and Knowledge Technology (IKT)*, pages 1–7. IEEE, 2020.

[101] Meenakshi Vanmali, Mark Last, and Abraham Kandel. Using a neural network in the software testing process. *International Journal of Intelligent Systems*, 17(1):45–62, 2002.

[102] KK Aggarwal, Yogesh Singh, Arvinder Kaur, and OP Sangwan. A neural net based approach to test oracle. *ACM SIGSOFT Software Engineering Notes*, 29(3):1–6, 2004.

[103] Seyed Reza Shahamiri, Wan Mohd Nasir Wan Kadir, and Suhaimi bin Ibrahim. An automated oracle approach to test decision-making structures. In *2010 3rd International Conference on Computer Science and Information Technology*, volume 5, pages 30–34. IEEE, 2010.

[104] Seyed Reza Shahamiri, Wan MN Wan-Kadir, Suhaimi Ibrahim, and Siti Zaiton Mohd Hashim. Artificial neural networks as multi-networks automated test oracle. *Automated Software Engineering*, 19:303–334, 2012.

[105] Husheng Zhou, Wei Li, Yuankun Zhu, Yuqun Zhang, Bei Yu, Lingming Zhang, and Cong Liu. Deepbillboard: Systematic physical-world testing of autonomous driving systems.

[106] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars.

[107] T. Y. Chen, S. C. Cheung, and S. M. Yiu. Metamorphic testing: A new approach for generating next test cases.

[108] T.Y. Chen , F.C. Kuo , H. Liu, P.L. Poon, D. Towey, T.H. Tse, Z.Q. Zhou. Metamorphic testing: A review of challenges and opportunities. In *ACM Computing Surveys (CSUR) 51.1*, pages 1–27, 2018.

[109] Christian Murphy. *Metamorphic testing techniques to detect defects in applications without test oracles*. Columbia University, 2010.

[110] Sergio Segura Rueda, Dave Towey, Zhi Quan Zhou, and Tsong Yueh Chen. Metamorphic testing: Testing the untestable. *IEEE Software, 37 (3), 46-53.*, 2020.

[111] Jie M Zhang, Mark Harman, Lei Ma, and Yang Liu. Machine learning testing: Survey, landscapes and horizons. *IEEE Transactions on Software Engineering*, 2020.

[112] Christian Murphy, Gail E. Kaiser, and Lifeng Hu. Properties of machine learning applications for use in metamorphic testing.

[113] Junhua Ding, Xiaojun Kang, and Xin-Hua Hu. Validating a deep learning framework by metamorphic testing. In Laura L. Pullum, editor, *Proceedings of the 2nd International Workshop on Metamorphic Testing*, ACM Digital Library, pages 28–34, Piscataway, NJ, 2017. IEEE Press.

[114] Xiaoyuan Xie, Joshua Ho, Christian Murphy, Gail Kaiser, Baowen Xu, and Tsong Yueh Chen. Application of metamorphic testing to supervised classifiers. *Proceedings. International Conference on Quality Software*, 2009(2009):135–144, 2010.

[115] Sadam Al-Azani and Jameleddine Hassine. Validation of machine learning classifiers using metamorphic testing and feature selection techniques. In Somnuk Phon-Amnuaisuk, Swee-Peng Ang, and Soo-Young Lee, editors, *Multi-disciplinary Trends in Artificial Intelligence*, pages 77–91, Cham, 2017. Springer International Publishing.

[116] Xiaoyuan Xie, Zhiyi Zhang, Tsong Yueh Chen, Yang Liu, Pak-Lok Poon, and Baowen Xu. Mettle: A metamorphic testing approach to assessing and validating unsupervised machine learning systems. *IEEE Transactions on Reliability*, 69(4):1293–1322, 2020.

[117] Zhi Quan Zhou, Shaowen Xiang, and Tsong Yueh Chen. Metamorphic testing for software quality assessment: A study of search engines. *IEEE Transactions on Software Engineering*, 42(3):264–284, 2016.

[118] Zhi Quan Zhou, ShuJia Zhang, Markus Hagenbuchner, T. H. Tse, Fei-Ching Kuo, and T. Y. Chen. Automated functional testing of online search services. *Software Testing, Verification and Reliability*, 22(4):221–243, 2012.

[119] Zhi Quan Zhou, T. H. Tse, F. C. Kuo, and T. Y. Chen. Automated functional testing of web search engines in the absence of an oracle. *Department of Computer Science, The University of Hong Kong, Tech. Rep. TR-2007-06*, 2007.

[120] Sergio Segura, José A. Parejo, Javier Troya, and Antonio Ruiz-Cortés. Metamorphic testing of restful web apis. In *Proceedings of the 40th International Conference on Software Engineering*, page 882, 2018.

[121] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. Deeptest: Automated testing of deep-neural-network-driven autonomous cars.

[122] Zeyu Sun, Jie M. Zhang, Mark Harman, Mike Papadakis, and Lu Zhang. Automatic testing and improvement of machine translation.

[123] Arlinta Christy Barus, Tsong Yueh Chen, Fei-Ching Kuo, Huai Liu, and Heinz W. Schmidt. The impact of source test case selection on the effectiveness of metamorphic testing. In *Proceedings of the 1st International Workshop on Metamorphic Testing*, ACM Digital Library, pages 5–11, New York, NY, 2016. ACM.

[124] Huai Liu, Fei-Ching Kuo, Dave Towey, and Tsong Yueh Chen. How effectively does metamorphic testing alleviate the oracle problem? *IEEE Transactions on Software Engineering*, 40(1):4–22, 2014.

[125] Tom Badgett, Glenford J Myers, et al. *The Art of Software Testing*. Wiley Online Library, 2023.

[126] Arlinta Christy Barus et al. *An in-depth study of adaptive random testing for testing program with complex input types*. PhD thesis, Ph. D. Thesis. Faculty of Information and Communication Technologies . . . , 2010.

[127] Sergio Segura, Gordon Fraser, Ana B. Sanchez, and Antonio Ruiz-Cortes. A survey on metamorphic testing. *IEEE Transactions on Software Engineering*, 42(9):805–824, 2016.

[128] Andreas Zimmermann, Andreas Lorenz, and Reinhard Oppermann. An operational definition of context. In *Modeling and Using Context: 6th International and Interdisciplinary Conference, CONTEXT 2007, Roskilde, Denmark, August 20-24, 2007. Proceedings 6*, pages 558–571. Springer, 2007.

[129] Claudia Nobis and Tobias Kuhnimhof. *Mobilität in Deutschland – MiD: Ergebnisbericht*. 2018.

[130] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):1–48, 2019.

[131] Santiago Matalonga, Felyppe Rodrigues, and Guilherme Horta Travassos. Characterizing testing methods for context-aware software systems: Results from a quasi-systematic literature review. *Journal of Systems and Software*, 131:1–21, 2017.

[132] Thomas G Dietterich and Eun Bae Kong. Machine learning bias, statistical bias, and statistical variance of decision tree algorithms. Technical report, Citeseer, 1995.

[133] Ken Perlin. Improving noise. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 681–682, 2002.

[134] Wensheng Dong, Xinyan Zhang, and Caijun Zhang. Generation of cloud image based on perlin noise. In *2010 International Conference on Multimedia Communications*, pages 61–63. IEEE, 2010.

Betreute Bachelor- und Masterarbeiten

[Boy17] Boyan Boychev. *Konzeptionierung und Modellierung der Elektronik-Architektur eines Demonstrators für zukünftige intelligente Fahrzeuge*. Bachelorarbeit, Karlsruher Institut für Technologie (KIT), 2017.

[Che20] Sihan Chen. *Ein Ansatz für Maschinelles Lernen in der Produktion: Ein Software-Framework zur Ausführung von künstlichen Neuronalen Netzen in C++*. Masterarbeit, Karlsruher Institut für Technologie (KIT), 2020.

[Che21] Yalin Chen. *Kontinuierliches Lernen von tiefen neuronalen Netzen für Anwendungen mit inkrementellen Daten*. Masterarbeit, Karlsruher Institut für Technologie (KIT), 2021.

[Dja20] Aulia Jana Djamal. *Konzeptioneller Vergleich und Bewertung verschiedener Strategien zur automatisierten Korrektur von Fehlern im Flechtprozess*. Bachelorarbeit, Karlsruher Institut für Technologie (KIT), 2020.

[Duo21] Martin Duong. *Synthetische Datengenerierung und Metamorphes Testen eines selbstlernenden Komfortsystems*. Bachelorarbeit, Karlsruher Institut für Technologie (KIT), 2021.

[Gai19] Moritz Gaiser. *Evaluation der Eignung von Deep Reinforcement Algorithmen für das autonome Fahren*. Masterarbeit, Karlsruher Institut für Technologie (KIT), 2019.

[Hah20] Celia Hahn. *Vergleich eines End-to-End Ansatzes für autonomes Fahren in den Simulationsumgebungen Carla und Airsim*. Bachelorarbeit, Karlsruher Institut für Technologie (KIT), 2020.

[Hur20] You Hur. *Anwendung von End-to-End-Lernen: Imitationslernen eines Demonstrationsfahrzeugs am Beispiel eines autonomen Überholassistenten*. Masterarbeit, Karlsruher Institut für Technologie (KIT), 2020.

[Jun20] Carolin Junk. *Vergleich von Reinforcement Learning und Modellprädiktiver Regelung zur Klimaregelung in Elektrobussen*. Masterarbeit, Karlsruher Institut für Technologie (KIT), 2020.

[KB20] Valentin Khan-Blouki. *Konzeptionierung eines kamerabasierten Deep Learning Systems zur automatisierten Vermessung kardiovaskulärer Implantate mit dem Ziel einer Korrektur des Flechtprozesses*. Bachelorarbeit, Karlsruher Institut für Technologie (KIT), 2020.

[Kep18] Felix Keppler. *Videobasierte Fahrmanöver zur Erweiterung der Auswertemethodik im Rahmen mobiler Emissionsmessungen*. Masterarbeit, Karlsruher Institut für Technologie (KIT), 2018.

[Kha19] Houssem Eddine Khatrchi. *Intelligente visuelle Inspektion von Flechtmustern durch maschinelles Lernen zur Fehlererkennung*. Masterarbeit, Karlsruher Institut für Technologie (KIT), 2019.

[Klu21] Simon Klug. *Lernerzentriertes Code-Assessment - Ein NLP-Ansatz für besseres Lernen und Lehren*. Masterarbeit, Karlsruher Institut für Technologie (KIT), 2021.

[Kru18] Fabian Krug. *Entwicklung eines Konzepts zur Transformation einer Rewardfunktion des Reinforcement Learning aus der Simulation in die Realität im Kontext des autonomen Fahrens*. Bachelorarbeit, Karlsruher Institut für Technologie (KIT), 2018.

[Kum22] Neeti Kumari. *Szenariobasiertes metamorphes Testen zur Evaluation einer selbstlernenden Funktion*. Masterarbeit, Karlsruher Institut für Technologie (KIT), 2022.

[Li19] Xing Li. *Learning from Demonstration for Trajectory Planning in Automated Driving*. Masterarbeit, Karlsruher Institut für Technologie (KIT), 2019.

[Mai20] Andreas Mai. *Visuelle Anomalieerkennung mittels Autoencodern*. Bachelorarbeit, Karlsruher Institut für Technologie (KIT), 2020.

[Mül20] Simon Müller. *Adaptive Gesture Control for Automotive Applications with Few-Shot-Learning*. Masterarbeit, Karlsruher Institut für Technologie (KIT), 2020.

[Mus22] Franz Muszarsky. *Erzeugung von synthetischen Daten mit Hilfe von „Generative Adverserial Nets“*. Bachelorarbeit, Karlsruher Institut für Technologie (KIT), 2022.

[Müt20] Ferdinand Mütsch. *TalkyCars: A Distributed Software Platform for Cooperative Perception among Connected Autonomous Vehicles based on Cellular-V2X Communication*. Masterarbeit, Karlsruher Institut für Technologie (KIT), 2020.

[Ngu19] Xuan Vinh Nguyen. *Konzeptionierung und Implementierung einer Methodik zur automatisierten Identifikation von emissionsträchtigen Fahrmanövern im Kontext des RDE-Prüfverfahrens*. Masterarbeit, Karlsruher Institut für Technologie (KIT), 2019.

[Oue20] Chaden Ouertani. *Anomalieerkennung in Zeitserien durch Convolutional Autoencoder*. Bachelorarbeit, Karlsruher Institut of Technology (KIT), 2020.

[Rau19] Vinzent Rau. *Machine Learning zur spektralen Rekonstruktion und Klassifikation*. Bachelorarbeit, Karlsruher Institut für Technologie (KIT), 2019.

[Sch20] Felix Schorle. *Automatische Sichtprüfung von kardiovaskulären Implantaten mit Neuronalen Netzen*. Bachelorarbeit, Karlsruher Institut für Technologie (KIT), 2020.

[Sch23] Felix Schorle. *Generierung von Test- und Simulationsumgebungen aus Fahrzeug-Diagnose Abläufen*. Masterarbeit, Karlsruher Institut für Technologie (KIT), 2023.

[Stü20] Valentin Stümpert. *Untersuchung und Vergleich von Methoden der Hyperparameter-Optimierung für verschiedene Klassen von neuronalen Netzen*. Bachelorarbeit, Karlsruher Institut für Technologie (KIT), 2020.

[Tre22] Dimitar Trendafilov. *Erklärung von AI-Modellen durch Simplifizierung*. Bachelorarbeit, Karlsruher Institut für Technologie (KIT), 2022.

[Ulu22] Umut Ege Ulucay. *Verbesserte Verständlichkeit beim Testen von KI-basierten Systemen*. Masterarbeit, Karlsruher Institut für Technologie (KIT), 2022.

[Zip18] Matthias Zipp. *Entwurf und Realisierung eines Systems zum Testen von maschinellen Lernverfahren auf Basis einer mobilen Plattform*. Bachelorarbeit, Karlsruher Institut für Technologie (KIT), 2018.

Eigene Veröffentlichungen

[BGH+19] Jürgen Becker, Daniel Grimm, Tim Hotfilter, Christopher Meier, Gabriela Molinar, Marco Stang, Simon Stock, and Wilhelm Stork. The QUA^{3}CK machine learning development process and the laboratory for applied machine learning approaches (LAMA). In *Symposium Artificial Intelligence for Science, Industry and Society (AISIS 2019), Mexiko-Stadt, Mexiko*, 2019.

[BLS+20] Martin Böhme, Andreas Lauber, Marco Stang, Luyi Pan, and Eric Sax. Using machine learning to optimize energy consumption of hvac systems in vehicles. In *Human Interaction and Emerging Technologies: Proceedings of the 1st International Conference on Human Interaction and Emerging Technologies (IHIET 2019), August 22-24, 2019, Nice, France*, pages 706–712. Springer, 2020.

[BSMS20] Martin Boehme, Marco Stang, Ferdinand Mütsch, and Eric Sax. Talkycars: A distributed software platform for cooperative perception. In *2020 IEEE Intelligent Vehicles Symposium (IV)*, pages 701–707. IEEE, 2020.

[GSNS21] Maria Guinea, Marco Stang, Irma Nitsche, and Eric Sax. Acceptance of smart automated comfort functionalities in vehicles. In *Human Interaction, Emerging Technologies and Future Applications IV: Proceedings of the 4th International Conference on Human Interaction and Emerging Technologies: Future Applications (IHIET–AI 2021), April 28-30, 2021, Strasbourg, France 4*, pages 331–338. Springer, 2021.

[GSS21] Daniel Grimm, Marco Stang, and Eric Sax. Context-aware security for vehicles and fleets: A survey. *IEEE Access*, 9:101809–101846, 2021.

[HSKBS21] Benedikt Haas, Marco Stang, Valentin Khan-Blouki, and Eric Sax. Introduction of an algorithm based on convolutional neural

networks for an automated online correction of braided cardiovascular implants. In *Human Interaction, Emerging Technologies and Future Applications IV: Proceedings of the 4th International Conference on Human Interaction and Emerging Technologies: Future Applications (IHIET–AI 2021), April 28-30, 2021, Strasbourg, France 4*, pages 28–36. Springer, 2021.

[KJSS23] David Kraus, Carolin Junk, Marco Stang, and Eric Sax. Context-aware policy for route planning and feasible vehicle technologies. In *2023 IEEE International Automated Vehicle Validation Conference (IAVVC)*, pages 1–6. IEEE, 2023.

[SBS19] Marco Stang, Martin Böhme, and Eric Sax. Applied machine learning: Reconstruction of spectral data for the classification of oil-quality levels. *The Eurasia Proceedings of Science Technology Engineering and Mathematics*, 2019. ISSN: 2602-3199.

[SBS+20] Simon Stock, Alain Bertemes, Marco Stang, Martin Böhme, Daniel Grimm, and Wilhelm Stork. Feedi - a smart wearable foot-band for navigation and guidance using haptic feedback. In *Human Interaction, Emerging Technologies and Future Applications II: Proceedings of the 2nd International Conference on Human Interaction and Emerging Technologies: Future Applications (IHIET–AI 2020), April 23-25, 2020, Lausanne, Switzerland*, pages 349–355. Springer, 2020.

[SGGS20] Marco Stang, Daniel Grimm, Moritz Gaiser, and Eric Sax. Evaluation of deep reinforcement learning algorithms for autonomous driving. In *2020 IEEE intelligent vehicles symposium (IV)*, pages 1576–1582. IEEE, 2020.

[SMRS20] Marco Stang, Christopher Meier, Vinzenz Rau, and Eric Sax. An evolutionary approach to hyper-parameter optimization of neural networks. In *Human Interaction and Emerging Technologies: Proceedings of the 1st International Conference on Human Interaction and Emerging Technologies (IHIET 2019), August 22-24, 2019, Nice, France*, pages 713–718. Springer, 2020.

[SMS21] Marco Stang, Maria Guinea Marquez, and Eric Sax. CAGEN-context-action generation for testing self-learning functions. In *Human Interaction, Emerging Technologies and Future Applications IV: Proceedings of the 4th International Conference on*

Human Interaction and Emerging Technologies: Future Applications (IHIET–AI 2021), April 28-30, 2021, Strasbourg, France 4, pages 12–19. Springer, 2021.

[SSB+21] Marco Stang, Martin Sommer, Daniel Baumann, Yuan Zijia, and Eric Sax. Adaptive customized forward collision warning system through driver monitoring. In *Proceedings of the Future Technologies Conference (FTC) 2020, Volume 2*, pages 757–772. Springer, 2021.

[SSKS23] Marco Stang, Martin Sommer, David Kraus, and Eric Sax. Improving the validation of automotive self-learning systems through the synergy of scenario-based testing and metamorphic relations. In *Proceedings of the IEEE/ACM 10th International Conference on Big Data Computing, Applications and Technologies*, pages 1–7, 2023.

[SSM+22] Marco Stang, Simon Stock, Simon Müller, Eric Sax, and Wilhelm Stork. Development of a self-learning automotive comfort function: an adaptive gesture control with few-shot-learning. In *2022 International Conference on Connected Vehicle and Expo (ICCVE)*, pages 1–8. IEEE, 2022.

[SSS23] Marco Stang, Marc Schindewolf, and Eric Sax. Unravelling scenario-based behaviour of a self-learning function with user interaction. *Human Interaction & Emerging Technologies (IHIET 2023): Artificial Intelligence & Future Applications*, 111(111), 2023.

[SSVS24] Marco Stang, Luca Seidel, Veljko Vučinić, and Eric Sax. Exploring metamorphic testing for self-learning functions with user interactions. *Human Interaction and Emerging Technologies (IHIET-AI 2024): Artificial Intelligence and Future Applications*, 120(120), 2024.

[VSSS24] Veljko Vučinić, Luca Seidel, Marco Stang, and Eric Sax. Usid-unsupervised identification of the driver for vehicle comfort functions. *Human Interaction and Emerging Technologies (IHIET-AI 2024): Artificial Intelligence and Future Applications*, 120(120), 2024.