Bachelorarbeit gemäß § 17 der Allgemeinen Prüfungsordnung vom 09.02.2023
im Bachelorstudiengang Wirtschaftsinformatik
an der Hochschule für angewandte Wissenschaften Neu-Ulm

Bachlorarbeit

Generative Künstliche Intelligenz in der AGV-Testautomatisierung

Entwicklung eines LLM-basierten Chatbots zur Erstellung von AGV-Tests mit visueller Unterstützung

zur Erlangung des akademischen Grades

Bachelor of Science (B. Sc.)

im Studiengang

Wirtschaftsinformatik

Erstprüfer	Prof. Dr. Jens Kolb
Zweitprüfer	Prof. Dr. Alexander Bartel
Betreuer	Paul Wolf
Vorgelegt von	Niklas Seemann, 3138963

Thema erhalten:	01.10.2024
Arbeit abgeliefert:	31.01.2025

Eidesstattliche Erklärung

Ich erkläre hiermit an an Eides statt, dass ich die vorliegende Bachlorarbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und noch nicht veröffentlicht.

Neu-Ulm, den 31.01.2025

Niklas Seemann

Abstract

Die vorliegende Arbeit untersucht die Entwicklung eines Systems zur Erstellung von Testsequenzen für Automated Guided Vehicles durch den Einsatz eines Chatbots und generativer Künstlicher Intelligenz. Das Ziel ist es, die Effizienz und Genauigkeit bei der Erstellung von Testsequenzen zu verbessern und diese auch für Benutzer ohne tiefgehende Vorkenntnisse zugänglich zu machen.

Im Rahmen der Arbeit wird ein auf einem Large Language Model basierender Chatbot entwickelt, der durch den gesamten Prozess der Test- und Kartenerstellung führt. Der Chatbot nutzt einen modularen Ansatz mit spezialisierten Agenten für allgemeine Fragen, Karten- und Testanpassungen. Die Verlässlichkeit und Benutzerfreundlichkeit des Systems werden durch eine testbasierte Evaluation und Expertenbefragungen untersucht.

Die Ergebnisse zeigen eine bereinigte Erfolgsquote von 92,78 % bei der Erstellung und Anpassung von Gherkin-Tests und Karteninformationen. Bei den Expertenbefragungen mit 10 Teilnehmern ergab sich eine durchschnittliche Bewertung von 4,65 von 5 Punkten, was auf eine hohe Zufriedenheit und Benutzerfreundlichkeit hinweist. Damit bietet der entwickelte Chatbot eine vielversprechende Lösung zur Unterstützung bei der Erstellung und Anpassung von Gherkin-Tests und Karteninformationen für Automated Guided Vehicles.

Keywords

GenAI, LLM, Gherkin, Chatbot, BDD, AGV

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listingverzeichnis

Abkürzungsverzeichnis

AGV	Automated Guided Vehicle
vAGV	virtual Automated Guided Vehicle
LIF	Layout Interchange Format
GenAI	Generative Künstliche Intelligenz
KI	Künstliche Intelligenz
LLM	Large Language Model
NLP	Natural Language Processing
NLU	Natural Language Understanding
NLG	Natural Language Generation
GPT	Generative Pre-trained Transformer
BERT	Bidirectional Encoder Representations from Transformers
BDD	Behavior Driven Development
TDD	Test Driven Development
CLI	Comand Line Interface

1 Einleitung

Generative Künstliche Intelligenz (GenAI) hat in den letzten Jahren nicht nur Einfluss auf unterschiedliche Bereiche des alltäglichen Lebens durch Systeme wie ChatGPT gewonnen, sondern auch die Wirtschaft und Industrie verändert. Die rasante Entwicklung von GenAI bietet neue, innovative Ansätze zur Automatisierung und Effizienzsteigerung in verschiedenen Bereichen, darunter die Automatisierung von Testprozessen für Automated Guided Vehicles (AGVs), die in modernen Produktions- und Logistiksystemen eine zentrale Rolle spielen.

Diese Arbeit widmet sich der Entwicklung eines Chatbot-gestützten Systems zur Erstellung von AGV-Testsequenzen durch eine Frage-Antwort-Kommunikation. Ergänzt wird das System durch GenAI mit einem besonderen Fokus auf ein Large Language Model (LLM) und visuelle Hilfsmittel, wie einer Layout Interchange Format (LIF)-Karte mit den verfügbaren, abfahrbaren Positionen und Routen. Das Forschungsinteresse konzentriert sich hierbei auf die Frage, inwieweit ein solcher Ansatz die Effizienz und Genauigkeit bei der Erstellung von Testsequenzen verbessern kann, woraus sich die zwei folgenden Forschungsfragen ergeben:

Q1 Wie verlässlich können mithilfe eines LLM-basierten Chatbots valide Gherkin-Tests und LIF-Karteninformationen für AGVs anhand von Chat-Anweisungen erstellt und modifiziert werden?

Q2 Wie benutzerfreundlich ist die Erstellung und Anpassung von Gherkin-Tests und LIF-Karteninformationen für AGVs mithilfe eines LLM-basierten Chatbots?

Das Ziel dieser Arbeit ist es, ein funktionales System zu entwickeln, das durch einen Chatbot gesteuert wird und mithilfe von GenAI AGV-Testsequenzen und Karteninformationen generiert. Dabei soll untersucht werden, wie gut der Chatbot Benutzeranweisungen interpretieren und korrekte Testsequenzen und Karteninformationen erstellen kann. Zudem wird die Benutzerfreundlichkeit des Systems im Rahmen einer Expertenbefragung evaluiert.

Der Forschungsstand zeigt, dass Chatbots und GenAI bereits in verschiedenen Bereichen erfolgreich eingesetzt werden, um repetitive Aufgaben zu automatisieren und komplexe Prozesse zu erleichtern. Insbesondere im Bereich der Softwareentwicklung haben sich GenAI-Modelle als praktische Werkzeuge erwiesen. Die (teil-)automatisierte Erstellung von Tests im Rahmen von Behavior Driven Development (BDD) beschäftigt unterschiedliche Bereiche der Forschung. Ein spezifisches Beispiel dafür ist die Arbeit von Soeken et al., die

eine (teil-)automatisierte Methode zur Erstellung von Testszenarien mit Natural Language Processing (NLP) untersucht haben.

Die Relevanz dieses Themas liegt in der derzeitigen Weiterentwicklung eines Steuersystems für AGVs sowie der wachsenden Bedeutung von AGVs in der Produktion. Die Materialbelieferung einzelner Stationen stellt dabei einen systemkritischen Aspekt dar, durch welchen die Produktion zum Stillstand kommen kann. Aus diesem Grund gewinnt ausführliches Testen neuer Funktionen und Routen zunehmend an Wichtigkeit. Das Erstellen solcher Tests erfordert bislang unter anderem tiefgehendes Wissen über die internen Abhängigkeiten verfügbarer Testschritte. Durch die Entwicklung des Chatbot-gestützten Systems sollen diese Hürden minimiert und somit mehr Benutzern die Möglichkeit eröffnet werden, Tests zu schreiben und das AGV-System kontinuierlich weiterzuentwickeln.

Die Methodik dieser Arbeit umfasst die Entwicklung und Implementierung eines Chatbot-gestützten Systems unter Verwendung von GenAI-Modellen. Zunächst wird eine Anforderungsanalyse durchgeführt, gefolgt von der Konzeption und Implementierung des Systems. Anschließend wird eine Validierung durch Tests und Expertenbefragungen durchgeführt, um die Effizienz und Benutzerfreundlichkeit des Systems zu bewerten.

Diese Arbeit ist in mehrere Kapitel gegliedert. Kapitel 2 zu den theoretischen Grundlagen, in dem die relevanten Konzepte und Technologien erläutert werden. Kapitel 3 beschreibt das Konzept des Systems. In Kapitel 4 wird der bestehende Forschungsstand untersucht. Kapitel 5 zeigt die Implementierung mit den dabei aufgetretenen Herausforderungen. Kapitel 6 umfasst die Validierung der durchgeführten Tests und Expertenbefragungen. Abschließend werden in Kapitel 7 die Ergebnisse diskutiert und in Kapitel 8 eine Zusammenfassung sowie ein Ausblick auf mögliche zukünftige Entwicklungen gegeben.

2 Grundlagen

In diesem Kapitel wird auf grundlegende Konzepte und Technologien eingegangen, die für das Verständnis der nachfolgenden Kapitel vorausgesetzt werden. Zunächst wird hierbei auf GenAI eingegangen, was ein LLM ist und Beispiele für deren Verwendung gegeben. Anschließend wird auf NLP eingegangen, die Grundlagen zu BDD erklärt und abschließend das verwendete Kartenformat beschrieben.

2.1 Generative Künstliche Intelligenz

GenAI unterscheidet sich von der diskriminativen Künstliche Intelligenz (KI), die sich auf die Erkennung von Mustern oder Vorhersagen konzentriert, durch die Fähigkeit neue Inhalte erzeugen zu können [1, S. 550f.]. In Folge fallender Preise für Rechenleistung und wachsender Verfügbarkeit von Open-Source-Lösungen sanken die Kosten für GenAI und somit stieg die Zugänglichkeit für eine Vielzahl an Nutzern und Entwicklern [1, S. 551]. Die Erzeugung von neuen Inhalten in unterschiedlichen Formaten wie Text, Bild, Video oder Audio liefert neue, zuvor nie dagewesene Möglichkeiten, wie mit Maschinen interagiert und gearbeitet werden kann. Neben den Vorteilen und dem Potenzial dieser Technologie zeichnen sich auch neue Herausforderungen und Risiken bei der Verwendung ab. Die Generierung von neuen Inhalten birgt beispielsweise die Gefahr, dass der Inhalt durch verzerrte Informationen verfälscht und somit nicht vertrauenswürdig ist [1, S. 550ff.], [2, S.21].

GenAI-Modelle können auf unterschiedliche Arten klassifiziert werden, beispielsweise auf Basis der Architektur oder der verschiedenen Ein- und Ausgabeformate. Die bekanntesten Ein- und Ausgabeformate sind Text, Bild, Musik, Sprache und Audio. Diese Formate können in unterschiedlichsten Kombinationen in Modellen enthalten sein, um beispielsweise mit einer textuellen Beschreibung ein Bild zu erstellen oder ein bestehendes Bild zu erklären [3, S. 25ff.]. Da sich diese Arbeit auf die Ein- und Ausgabe in Textform beschränkt, wird auf die anderen Formate nicht weiter eingegangen. Zudem wird in dieser Arbeit die Transformer-Architektur verwendet, welche im Folgenden genauer beschrieben wird [3, S. 10ff.]:

Transformers bestehen aus mehreren sogenannten Selbstaufmerksamkeitsschichten und Feed-Forward-Neuronalen Netzwerkschichten. Diese Selbstaufmerksamkeitsschichten dienen dazu, Gewichte zwischen allen Paaren der Eingabekomponenten zu berechnen, wodurch es dem Modell ermöglicht wird, verschiedene Teile der Eingabesequenz zu untersuchen und relevante Informationen zu berücksichtigen. Die Feed-Forward-Schicht verbessert durch nicht

lineare Transformationen die Fähigkeit, komplexe Muster und Beziehungen in den Daten zu erfassen. Die Erfassung des Kontextes ist essenziell wichtig, wenn es um die Verarbeitung natürlicher Sprache geht, da dieselben Wörter in unterschiedlichen Kontexten unterschiedliche Bedeutungen haben können. Ein Beispiel für diesen Einfluss wäre das Wort "Bank", das sowohl im Kontext des Sitzens auf einer Bank als auch des Geldabhebens auf einer Bank verwendet werden kann. Zusätzlich ist das Trainieren von Transformers mit unstrukturierten Daten möglich. Die Kombination dieser Merkmale macht die Transformers-Architektur zu einem weit verbreiteten und effizienten Design in der natürlichen Sprachverarbeitung und Textkategorisierung [3, S. 18ff.],[4, S. 27f.],[5, S. 357ff.], [6, S. 5ff.].

Zwei bekannte Modelltypen, die dem Transformers-Design folgen sind:

- **Bidirectional Encoder Representations from Transformers (BERT)** ist ein von Google entwickeltes Sprachmodell, das aus mehreren Ebenen bidirektionaler Transformer-Encoder besteht. BERT maskiert während des Vortrainings zufällig Wörter in der Eingabesequenz. Dies sind etwa 15 % der Tokens, die durch ein spezielles Token <mask> ersetzt werden. Das Modell wird darauf trainiert, die fehlenden Tokens an den entsprechenden Stellen vorherzusagen. Die bidirektionale Eigenschaft beschreibt die Tatsache, dass die Wörter vor und nach der Markierung betrachtet und als Informationsquelle für die Vorhersage des passenden Wortes herangezogen werden. BERT kann durch überwachte Lernmethoden auf spezifische Anwendungsfälle angepasst werden [3, S. 19], [4, S. 29], [5, S. 390ff.].
- **Generative Pre-trained Transformer (GPT)** ist ein 2018 von OpenAI entwickeltes Modell, das laut Angaben von OpenAI neue Möglichkeiten des Finetunings mithilfe von Eingabetransformationen bot. Im Gegensatz zu BERT ist GPT ein reines Decodermodell wobei das Zero-Shot-Lernen durch die im Gegensatz zu überwachten Lernmethoden höhere Effizienz durch das in der Regel wegfallende manuelle organisieren der Trainingsdaten eine wichtige Innovation darstellt. Unter Zero-Shot-lernen versteht man ein Modell, das neue, noch nie zuvor gesehene, Konzepte und Objekte durch das aus riesigen Datenmengen gewonnene Wissen erkennen kann. Bereits 2019 entwickelte OpenAI das Nachfolgemodell GPT-2. Das neue Modell lieferte mit etwa 1,5 Milliarden Parametern und einem Datensatz von mehr als 8 Milliarden Dokumenten einen etwa 15-mal größeren Umfang als das erste GPT Modell [7], [8, S. 108], [4, S. 56], [5, S. 390ff.], [6, S. 31f.].

Das neuste Modell von OpenAI, GPT-4o ist ein sogenanntes multimodales-Modell welches neben Text auch Audio, Bild, Video und deren Kombinationen als Eingabe und Ausgabe ermöglicht. Der sogenannte „Knowledge Cutoff “ beschreibt den Zeitpunkt bis zu den Trainingsdaten gesammelt wurden und liegt im Oktober 2023. Die Kerndatensätze, die zum Trainieren verwendet wurden, waren öffentliche Websites, Code und mathematische Daten sowie multimodale Datensätze mit Text, Bild, Audio und Video. Diese Eigenschaften des Modells machen es in verschiedenen Anwendungsfällen nutzbar [9].

In den nachfolgenden Abschnitten wird auf die für die Arbeit wichtigen Elemente von GenAI wie dem LLM und Prompt Engineering, NLP sowie zwei bekannte Anwendungsbeispiele eingegangen.

2.1.1 Large Language Model

Ein LLM ist ein Sprachmodell, das auf einem umfangreichen Datensatz trainiert wurde. LLMs sind in der Lage Textinhalte zu verstehen und neue, von menschlichen Texten kaum zu unterscheidende, zu erzeugen. Die Funktionsweise dieser Modelle basiert auf dem Prinzip der probabilistischen Vorhersage. Dabei werden Muster und Abhängigkeiten in Wortfolgen erlernt, um die Wahrscheinlichkeit für das Auftreten des Wortes im gegebenen Kontext zu schätzen und somit das nächste Wort vorherzusagen. Durch die Erfassung von Regelmäßigkeiten in Sprachen sind die Modelle in der Lage, kontextbezogene Texte zu generieren und Konversationsfragen zu beantworten [3, S. 17f.], [10, S. 2630], [11, S. 41ff.].

In den letzten Jahren ist der Einsatz von LLMs in der Software-Entwicklung gestiegen. Die Möglichkeiten von Codegenerierung sowie dem Interagieren mit Code durch natürliche Sprache machen die Technologie besonders attraktiv [12, S. 112]. Allerdings bergen die im vorherigen Abschnitt erwähnten Herausforderungen und Risiken von GenAI auch Gefahren, die LLMs zu potenziellen Fehlerquellen machen. Die stetige Weiterentwicklung von Nicht-Open-Source-Modellen birgt das Risiko, dass sich bei jedem Update unerwartete Verhaltensänderungen der Modelle ergeben können. Ein weiteres Problem bei der Nutzung von LLMs ist die mangelnde Reproduzierbarkeit, da bei einigen Modellen trotz gleicher Eingabe unterschiedliche Ausgaben geliefert werden können [12, S. 103f.], [3, S. 29].

Nichtsdestotrotz sind LLMs eine bedeutende Entwicklung im Bereich des maschinellen Lernens der letzten Jahre. Die Größe dieser Modelle beläuft sich zum aktuellen Stand auf

etwa eine Billion (10^{12}) Parameter wodurch die Entwicklung und das hierin enthaltende Training hohe Hardwareanforderungen voraussetzt. Andererseits ist durch die steigende Leistungsfähigkeit und damit verbundenen breiten Einsatzmöglichkeiten der Modelle der Bedarf an Feinanpassungen geringer [5, S. 390ff.].

Im Rahmen der Verbreitung von LLMs ist ein neuer Bereich entstanden, der sich mit dem Formulieren der Eingabesequenzen dieser Modelle beschäftigt, genannt Prompt-Engineering. Die textbasierten Eingaben in das Modell stellen einen sogenannten Prompt dar, welcher die Anforderungen, Aufgaben oder die vom Modell zu beantwortende Frage beinhalten kann. Ein Prompt dient grundsätzlich dazu, dem Modell eine Richtung, Aufgabe und Orientierung zu liefern [5, S. 394], [3, S. 27]. Eine Auswahl von Prompt-Typen ist in Tabelle 1 zu sehen.

Prompt-Typ	**Beschreibung**	**Beispiel**
System	Wird häufig als Einstieg verwendet und enthält Informationen, wie eine Rolle, zur Bearbeitung	Du bist ein Experte für das Schreiben von Kurzgeschichten. Deine Aufgabe ist es ...
Instructive	Enthält eine spezifische Aufgabe, die bearbeitet werden soll.	Schreibe mir einen Aufsatz mit 200 Wörtern zum Thema Autofahren...
Contextual	Beschreibt eine Aufgabe und beinhaltet spezifische Informationen für die Bearbeitung	Schreibe mir einen Aufsatz über Autos. Beachte, dass hierbei nur Autos von Mercedes-Benz genannt werden sollen.
Question-Answer	Dient dazu, die Antwort auf eine spezifische Frage zu liefern.	Was sind die Schwierigkeiten bei der Erstellung personalisierter Angebote in der Automobilindustrie?

Tabelle 1: Typen von Prompts mit Beispielen und Beschreibung. Zu sehen sind die vier Prompt-Typen: System, Instructive, Contextual und Question-Answer.

Quelle: In Anlehnung an Giray [10, S. 2631].

Durch die exakte Beschreibung der Rolle, Hintergrundinformationen, Aufgabe und Ausgabeformat können häufige Fehler vermieden werden, die durch ungenaue Prompts herbeigeführt werden. Häufige Fehler sind die Verwendung von mehrdeutigen Wörtern, da diese regionsspezifisch unterschiedliche Bedeutungen haben und das Modell diese möglicherweise nicht wie erwartet interpretiert, das Fehlen von Informationen was dazu führt, dass Antworten des Modells schwammig und ungenau sind, sowie das Einbinden von Widersprüchen in den Prompt wodurch das Modell keine eindeutige Lösung liefern kann. Das Schreiben von guten Prompts ist für die Qualität der Ausgaben von LLMs von wesentlicher Bedeutung und stellt eine grundlegende Fähigkeit bei der Arbeit mit GenAI dar [10, S. 2631f.].

2.1.2 Anwendung mit GenAI

GenAI hat sich bereits in unterschiedlichen Bereichen des alltäglichen Lebens verbreitet, und dies ist nicht zuletzt auf das sehr bekannte System ChatGPT zurückzuführen. Jedoch ist auch in der Software-Entwicklung GenAI ein täglicher Begleiter durch Systeme wie GitHub Copilot. Im Folgenden werden die Systeme ChatGPT und GitHub Copilot vorgestellt und genauer beschrieben.

- **ChatGPT** ist ein von OpenAI am 30. November 2022 veröffentlichtes System, welches Nutzern von der Beantwortung von Fragen, dem Zusammenfassen von Inhalten bis hin zu zum Erzeugen von Bildern erlaubt GenAI zu verwenden. Bereits eine Woche nach der Einführung erreichte das System eine Million Anmeldungen. Zum Vergleich benötigte Facebook hierfür zehn Monate. Dieses Chat-basierte System unterscheidet sich von einer klassischen Suchmaschine, indem es menschenähnliche Interaktion per Chat ermöglicht. Dies und die einfache Verwendbarkeit machen ChatGPT zu einer breit genutzten Anwendung für das Schreiben und erklären von Texten oder dem Suchen und zusammenfassen von Inhalten [8, S. 11f.], [13, S. 277].
- **GitHub Copilot** ermöglicht Entwicklern das Arbeiten mit einem eigenen Assistenten der bei simplen Aufgaben als auch bei der Autovervollständigung im Code und der Fehlersuche unterstützt. Dies ist möglich, da GitHub Copilot auf eine große Anzahl an Open-Source-Projekten in unterschiedlichsten Programmiersprachen zugreifen kann. Eine Studie der Bilkent University aus der Türkei zeigte, das über 90 % der generierten Codeabschnitte gültig waren [14, S. 62]. Zudem erzielten Teilnehmer einer von GitHub durchgeführten Studie mithilfe von GitHub Copilot eine bis zu 55 % schnellere Bearbeitungszeit von Aufgaben gegenüber den Teilnehmern ohne GitHub Copilot [15].

Beide vorgestellten Anwendungen integrieren eine Chatbot-Komponente. Bei ChatGPT fungiert diese Chatbot-Komponente als zentrale Interaktionsoberfläche der Anwendung. Wobei GitHub Copilot die Chatbot-Komponente und Autovervollständigung direkt im Code kombiniert, um mit dem Benutzer zu interagieren. Ein Chatbot ist ein System, das es Menschen ermöglicht, über natürliche Sprache mit einem Computer zu interagieren. Chatbots haben sich in den letzten Jahren weiterentwickelt und in verschiedenen Bereichen, wie dem Tourismus oder Kundenservice beispielsweise zur Analyse des Benutzerverhaltens, etabliert [16, S. 377ff.], [17, S.1012], [18, S. 10232], [19, S. 75f.].

2.2 Natural Language Processing

Die Verarbeitung natürlicher Sprache, genannt NLP, ist ein meist auf maschinellem Lernen basierender Bereich der KI und Computerlinguistik. NLP beschäftigt sich mit dem Verstehen, Manipulieren und Erzeugen menschlicher Sprache durch einen Computer und schließt somit die Lücke zwischen menschlicher Kommunikation und maschinellem Verständnis. Hierbei wird umfassendes Wissen über die Verwendung menschlicher Sprache gesammelt, um die Verarbeitung durch den Computer zu ermöglichen. Es spielt eine wesentliche Rolle bei Chatbots, Sprachassistenten und weiteren Anwendungen, in denen die menschliche Sprache im Fokus steht [16, S. 377], [4, S. 16ff.], [8, S. 95ff.].

Die Verarbeitung von natürlicher Sprache kann in zwei Kategorien unterteilt werden:

- **Natural Language Understanding (NLU)** ist der Prozess, die menschliche, genannt natürliche Sprache durch Algorithmen so anzupassen und zu bereinigen, dass eine für den Computer verständliche Form erreicht wird. Für diese Vorverarbeitung gibt es unterschiedliche Ansätze wie das sogenannte Tokenisieren. Beim Tokenisieren wird der Text in unterschiedliche Komponenten, Token genannt, unterteilt. Ein Beispiel für eine solche Tokenisierung wäre die Umwandlung des Satzes: „Die Sonne scheint hell.“ in die folgenden Token: [„Die“, „Sonne“, „scheint“, „hell“, „.“]. Eine andere Technik, die auf der Tokenisierung aufbaut, sind N-Gramme. Diese Technik dient ebenfalls zum Extrahieren des Kontextes aus einem Text. Man unterscheidet verschiedene Arten von N-Grammen durch den Wert von „n“, welcher die Anzahl der zusammenhängenden Token angibt. Häufige N-Gramme sind die Unigramme (n = 1) und Bigramme (n = 2). Nachdem ein Text in Token unterteilt wurde, können N-Gramme gebildet werden. So wären beispielsweise die Unigramme für denselben Satz wie im vorherigen Beispiel: [„Die“, „Sonne“, „scheint“, „hell“, „.“] was derselben Ausgabe wie beim Tokenisieren entspricht und die Bigramme für diesen Satz: [(„Die“, „Sonne“), („Sonne“, „scheint“), („scheint“, „hell“), („hell“, „.“)]. Diese Unterteilung von Sätzen in ihre Bestandteile hilft der weiteren Analyse [16, S. 377], [20, S. 5] [4, S. 16ff.], [8, S. 95ff.].
- **Natural Language Generation (NLG)** beschreibt den Vorgang, wie Texte in natürlicher Sprache erstellt werden. Hierbei können die Texte unterschiedliche Formen annehmen und sowohl einzelne Sätze sowie mehrseitige Geschichten umfassen. Ein solcher Generierungsvorgang umfasst neben der Inhaltsplanung auch die Inhaltsbestimmung und Inhaltsrealisierung also das Umsetzen von Plan und Inhalt in eine lineare Abfolge von

Wörtern. NLG beinhaltet sowohl die Daten-zu-Text-Generierung, wie das Generieren von Text aus Bildern, als auch die Text-zu-Text-Generierung welche das Zusammenfassen, erweitern und umschreiben von Texten sowie die Beantwortung von Fragen umfasst [21, S. 2].

2.3 Behavior Driven Development

BDD ist eine agile Methode zur Software-Entwicklung, die es sowohl Entwicklern als auch Nicht-Entwicklern ermöglicht, einen Beitrag zur Entwicklung zu leisten. Das Ziel von BDD ist es, das Verhalten der Software in eine einfache, für jeden verständliche Sprache umzuwandeln. Dadurch wird die Kommunikationslücke zwischen Entwicklern und Geschäftspartnern geschlossen und die Interaktion gefördert. BDD wird oft als Verfeinerung von Test Driven Development (TDD) gesehen, wobei der Fokus von der Prüfung auf die Identifizierung des zu erwartenden Verhaltens gelegt wird. Bei TDD werden vor Beginn der Entwicklung von Entwicklern Tests in Code geschrieben. Diese Tests beschreiben alle Szenarien, die das System bestehen soll. Nach dem Schreiben eines Tests muss dieser zunächst fehlschlagen, da ansonsten ein bereits bestehender Test getestet würde. Das Problem bei TDD ist jedoch, dass es sich bei den Tests meist um sogenannte Unit-Tests handelt, welche beispielsweise einzelne Funktionen testen. Kleine Anpassungen im Code können dazu führen, dass neben dem zu bestehenden Test auch andere, bereits bestandene Tests fehlschlagen. Deshalb setzt BDD auf das Testen von Verhalten statt Struktur [22, S. 269f.], [23, S. 105ff.], [24, S. 88008ff.], [25, S. 23f.].

Das Verhalten der Software wird häufig in der domänenspezifischen Sprache Gherkin definiert und durch eine „Given-When-Then “-Form beschrieben. *GIVEN* drückt die Vorbedingung für das Starten eines Tests aus, *WHEN* beschreibt die Bedingung, das Ereignis oder das Verhalten und *THEN* zeigt abschließend das erwartete Ergebnis. Die Möglichkeit, mithilfe von Gherkin das Verhalten der Software zu beschreiben, macht Gherkin zu einem weit verbreiteten Instrument im Kontext von BDD. Ein weiteres beliebtes Tool bei der Automatisierung von Akzeptanztests ist das Cucumber Framework, welches die Sätze innerhalb der Szenarien parst, um eine „Given-When-Then“-Form in Code umzuwandeln [22, S. 271], [23, S. 107], [24, S. 88008ff.], [26, S. 1f.], [25, S. 24ff.], [27, S. 12].

Die Gherkin-Phrasen müssen zunächst in eine sogenannte Feature-Datei geschrieben werden. Um die Feature-Datei auszuführen, ist das Implementieren einer Schritt-Definition für

jeden verfügbaren Schritt notwendig. Eine solche Schritt-Definition ist in Listing 1 abgebildet, mit der dazugehörigen Feature-Datei in Listing 2. Bei einer Schritt-Definition handelt es sich um eine Funktion, die bei erfolgreichem Parsen des Gherkin-Schrittes ausgeführt werden soll. Hierbei werden die Parameter, die ebenfalls in natürlicher Sprache in der Gherkin-Phrase eingebaut sind, extrahiert [22, S. 271], [25, S. 24ff.].

```
@Given("the user {word} is online")
  fun user_is_online(name: String?) {
    println("The user $name is online")
  }
```

Listing 1: Beispiel einer Schritt-Definition für eine Given-Vorbedingung.

Quelle: Eigenes Listing.

```
Feature: User Online Status
  Scenario: User sends a message
    Given the user X is online
```

Listing 2: Beispiel für eine simple Feature-Datei mit einer Given Phrase.

2.4 Layout Interchange Format

Das LIF ist ein vom Verband Deutscher Maschinen- und Anlagenbau e.V., Fachverband Fördertechnik und Intralogistik, definiertes Format, das eine standardisierte Definition von Fahrstreckenlayouts für AGVs ermöglichen soll. Das Format ist angelehnt an die Struktur der VDA-5050-Schnittstelle[1] zur Kommunikation zwischen den AGVs und der Leitsteuerung. LIF hat das Ziel, die VDA-5050-Schnittstelle durch Entkopplung zwischen AGV-Herstellern und Leitsteuerungsanbietern zu ergänzen. Bei LIF wird eine JSON-Struktur verwendet, was das Hinzufügen von Kartenabschnitten ohne weitere Anpassungen der restlichen Kartendefinition erlaubt [29, S. 4ff.].

Die Kartendefinition beginnt zunächst mit allgemeinen Meta-Informationen wie einer Projektidentifikation, dem Ersteller oder der verwendeten LIF-Version. Anschließend beginnt der Abschnitt, welcher das Layout der Karte festlegt. Dieses Layout enthält ebenfalls Meta-Informationen, beispielsweise zur Layout-Version, gefolgt von den für diese Arbeit relevanten anzupassenden Elementen, wie den Knoten und den Kanten (siehe Abbildung 1). Die Kanten beschreiben zugleich die Verbindung zwischen zwei Knoten und die Richtung, in welche das Fahrzeug fahren kann. Eine Verbindung in beide Richtungen ist möglich, wenn es eine Kante von jedem Knoten zu dem nächsten sowie eine umgekehrte Kante gibt. Ein Beispiel einer LIF Layout-Definition als JSON-String mit einer möglichen dazugehörigen Darstellung ist in Abbildung 1 zu sehen [29, S. 11ff., S. 23ff.].

[1] Weitere Informationen zur VDA-5050-Schnittstelle sind der Empfehlung vom Verband für Automobilindustrie (VDA) [28] nachzulesen

```
"layouts": [
    {
        <<meta-data>>
        "nodes": [
            {
                "nodeId": "N1",
                "nodePosition": { "x": 0, "y": 0 }
            },
            {
                "nodeId": "N2",
                "nodePosition": { "x": 0, "y": 10 }
            },
        ],
        "edges": [
            {
                "edgeId": "N1-N2",
                "startNodeId": "N1",
                "endNodeId": "N2"
            }
        ]
    }
]
```

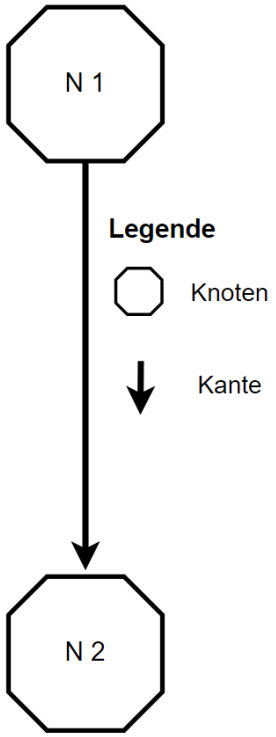

Abbildung 1: Zwei Knoten und eine Kante von Knoten 1 nach Knoten 2 und dem dazugehörigen Layout-Ausschnitt in JSON ohne Meta-Informationen.

Quelle: In Anlehnung an LIF [29, S. 26].

3 Konzept

In diesem Kapitel werden die Anforderungen an das System sowie deren Umsetzung und die Architektur des Chatbots beschrieben.

3.1 Anforderungen

Das entwickelte System muss zunächst in der Lage sein, mit dem Benutzer zu interagieren. Hierbei ist die GenAI-Komponente nützlich, um die Eingaben des Benutzers interpretieren zu können. Die Testsequenzen sollen schrittweise gestaltet werden, und das System muss in der Lage sein, bei Unklarheiten Rückfragen zu stellen. Zudem ist die Integration der virtuellen Karte ein wesentlicher Aspekt, um die genannten Ziele zu erreichen. Bei der virtuellen Karte ist es notwendig zu unterscheiden, ob entweder eine bestehende Karte oder eine speziell für diesen Test zu erstellende Karte gewählt werden soll. Im Falle der speziell für diesen Test zu erstellenden Karte soll diese parallel zur schrittweisen Testerstellung anpassbar und ebenfalls in der Anwendung erstellt werden können. Des Weiteren müssen die Testsequenzen gültig und ausführbar sein, da ansonsten das Parsen der Gherkin-Phrasen fehlschlägt. Abschließend sollen die generierten Testsequenzen exportierbar sein, um diese später in produktiven Tests ausführen zu können. Das System muss hinsichtlich der Testerstellung zudem in der Lage sein, definierte Vor- und Nachbedingungen selbstständig in den Test zu integrieren, allgemeine Testerstellungsregeln zu befolgen und neben einzelnen Schritten auch vollständige Szenarien anhand eines beschreibenden Szenario-Namens erstellen zu können. Weiter muss die Kartenerstellung sowohl durch das exakte Angeben von Positionen als auch durch das Beschreiben gewünschter Strukturen möglich sein.

Neben den genannten funktionalen Anforderungen ist die Benutzerfreundlichkeit und intuitive Testerstellung eine nicht-funktionale Anforderung. Der Benutzer soll möglichst wenig von den im Hintergrund laufenden Diensten und Anwendungen mitbekommen sowie auch ohne tiefergehendes Wissen über die Testerstellung im Projekt in der Lage sein, erste ausführbare Testsequenzen zu erstellen.

In den folgenden Abschnitten wird auf die Umsetzung der identifizierten Anforderungen Benutzerinteraktion, virtuelle Karte und das Erstellen gültiger Testsequenzen und Kartendaten eingegangen.

3.1.1 Interaktion mit dem Benutzer

Um die Interaktion mit dem Benutzer im System umzusetzen, soll ein Chatbot nach dem in Abschnitt 2.1.2 vorgestellten Beispiel zweier beliebter und populärer Anwendungen, ChatGPT und GitHub Copilot, entwickelt werden. Wie bereits in Abschnitt 2.1.2 erwähnt, ist ein Chatbot ein System, das die Interaktion zwischen einem Menschen und einem Computer über eine Frage-Antwort-Kommunikation ermöglicht. Dieser Chatbot stellt eine wesentliche Komponente des Systems dar und fungiert zugleich als die Interaktionsoberfläche des Benutzers mit dem System. Das Ziel ist, dass der Chatbot durch den gesamten Prozess der Karten- und Testerstellung führt, Vorschläge bietet und bei Unklarheiten Hilfestellungen liefert.

Die Fähigkeit, Benutzeranfragen zu interpretieren und kontextbezogene Antworten zu liefern, soll mithilfe eines LLM umgesetzt werden. Dieses LLM benötigt hierzu einen sogenannten System-Prompt (siehe Tabelle 1), welcher das Verhalten des LLMs sowie das zur Bearbeitung der Anfrage erforderliche Wissen definiert. Das erforderliche Wissen setzt sich aus der Gherkin-Struktur, den Karteninformationen und den generellen Regeln für die jeweilige Bearbeitung der Anfrage zusammen. Zudem ist das genannte Verhalten zu definieren, welches den Charakter und die vom Chatbot anzubietenden Funktionen und Fähigkeiten beinhaltet. Die Entwicklung des System-Prompts soll iterativ erfolgen und sich am Aufbau der konkreten Teilkomponenten, wie der Integration der Karte oder dem Testerstellungsprozess, orientieren.

3.1.2 Integration der virtuellen Karte

Wie bereits bei der Definition der Anforderungen erwähnt, teilt sich die Anforderung der Integration der virtuellen Karte zur Minimierung der Unübersichtlichkeit in zwei Varianten auf. Zu Beginn soll gewählt werden, ob eine neue Karte erstellt oder eine bereits bestehende Karte geladen werden soll. Diese Auswahl ermöglicht das Testen auf bereits bestehenden Produktionslayouts sowie das Testen einzelner Funktionen auf speziell hierfür erstellten Kartenabschnitten. Das Kartenformat basiert auf dem LIF, welches in Abschnitt 2.4 beschrieben wurde. Da in der virtuellen Variante der sich in der Weiterentwicklung befindenden Software, im Folgenden virtual Automated Guided Vehicle (vAGV)-Simulation genannt, bereits eine Lösung und Implementierung für das Rendern der Karte aus dem beschriebenen JSON-String existiert, wird dies in dieser Arbeit nicht weiter behandelt.

Das Erstellen oder Anpassen der Karteninformationen soll durch die Kommunikation mit dem Chatbot erfolgen. Der Chatbot stellt zielgerichtete Fragen, um die benötigten Informationen wie Positionen oder Richtungen der Kanten abzufragen. Darüber hinaus muss der Chatbot in der Lage sein, geeignete Rückfragen zu stellen, um bei Problemen zu unterstützen und entsprechende Hilfestellungen zu liefern. Nach jeder Anpassung der Karte muss der JSON-String an den korrekten Endpunkt gesendet werden. Abschließend ist die Integration der von der vAGV-Simulation erstellten grafischen Karte in die Benutzeroberfläche erforderlich.

3.1.3 Erstellen von gültigen Testsequenzen und Kartendaten

Das Erstellen von gültigen Testsequenzen ist eine grundlegende Anforderung an das System. Dies ist erforderlich, um diese erfolgreich parsen zu können. Um dies zu erreichen, benötigt es mehrere Schritte wie das Bereitstellen aller notwendigen Informationen über Gherkin, den Testerstellungsprozess sowie alle verfügbaren Knoten und Kanten. Der Prozess zur Erstellung soll ebenfalls durch den Chatbot geleitet werden. Hierbei soll der Test von einem LLM erstellt und anschließend von einem weiteren LLM überprüft werden. Da sich jedoch nicht vollständig auf die Antwort eines LLMs verlassen werden kann, ist eine zusätzliche Prüfung notwendig. Im Zuge dieser zusätzlichen Überprüfung soll der Parsing-Prozess durchgeführt werden, welcher Aufschluss darüber gibt, ob der erstellte Test zulässige Phrasen enthält. Hierbei soll das in Abschnitt 2.3 erwähnte Tool Cucumber sowie die vorgestellten Schritt-Definitionen zum Einsatz kommen. Das Ergebnis aus dieser letzten Überprüfung entscheidet darüber, ob der Test in der Benutzeroberfläche angezeigt oder eine Nachricht mit einer entsprechenden Fehlerbeschreibung an den Benutzer gesendet werden soll.

3.1.4 Exportieren der erstellten Testsequenzen

Die Anforderung, das Erstellte zu sichern und für eine spätere produktive Ausführung bereitzustellen, stellt den letzten Schritt in der Interaktion mit dem System dar. Hierbei müssen die Karte sowie die Feature-Datei extrahiert werden. Idealerweise soll dies über ein ZIP-Archiv geschehen, um die extrahierten Daten einander zuordnen zu können. Die Karte soll hierbei als JSON-Datei, welche den JSON-String der Kartendefinition enthält, abgespeichert werden. Dieses Paket soll stets exportierbar sein, um dem Benutzer die Möglichkeit der Datensicherung zu bieten.

3.2 Architektur

Für den Aufbau des Chatbots wurden zwei Ansätze untersucht. Zunächst wurde ein Ansatz betrachtet, bei dem lediglich ein großer System-Prompt für den allgemeinen Chat sowie die Erstellung der Karte und der Gherkin-Tests verwendet wird. Dieser große Prompt soll alle Informationen enthalten, die das System kennen muss, um mit dem Benutzer Tests und Karten erstellen zu können. Lediglich die Überprüfung und Anpassung der Karte in der Simulation wird von weiteren Komponenten übernommen.

Die Alternative zu diesem vereinfachten Ansatz ist das Aufteilen in Komponenten mit spezifischen Aufgaben. Dieser Ansatz, im Folgenden Agenten-Ansatz genannt, teilt die Funktionalität des ersten Ansatzes in drei Agenten auf, wobei die Auswahl des jeweiligen Agenten durch den sogenannten Mediator erfolgt. Der Mediator verfügt über ein LLM mit spezifischem Prompt zur Auswahl des richtigen Agenten basierend auf dem vorherigen Chatverlauf und der Anfrage des Benutzers. Der Mediator gibt anschließend die Entscheidung zurück, welche aus einer Liste der aufzurufenden Agenten mit der jeweiligen Aufgabe des Agenten besteht. Auf Basis dieser Entscheidung wird anschließend ein oder mehrere der folgenden Agenten aufgerufen:

- **Allgemeiner Agent:** Der erste Agent übernimmt das Antworten auf generelle Fragen, die keinen Bezug zu Gherkin oder der LIF-Karte haben. Zudem dient dieser Agent als Anlaufstelle, wenn der Benutzer Hilfe benötigt oder einzelne Komponenten der Benutzeroberfläche erklärt bekommen möchte
- **Karten-Agent:** Der zweite Agent ist für das Erstellen und Anpassen der Karte zuständig. Er besitzt das gesamte Wissen über die LIF-Karte und die Anbindung an die Simulationsschnittstelle, um Änderungen in der Karte direkt umzusetzen. Zusätzlich wird im Anschluss an die Kartenerstellung geprüft, ob die Änderungen in der Karte durch den Benutzer gewünscht wurden. Hierzu dient ebenfalls ein LLM, das die alten Karteninformationen mit den neuen vergleicht.
- **Gherkin-Agent:** Der dritte Agent beschäftigt sich mit dem Erstellen, Anpassen, Validieren und Erklären von Gherkin-Tests. Hierbei soll die gesamte Logik rund um Gherkin und die Testregeln in diesem Agenten zusammengefasst werden. Bei Aufruf des Agenten entscheidet dieser zunächst, ob der Benutzer eine Anpassung des Tests benötigt oder ob es sich um eine Frage bezüglich bereits bestehender Testschritte handelt.

Somit kann vermieden werden, dass der Test ungewollt angepasst wird und die Anzahl an LLM-Aufrufen verringert werden. Im Falle einer Anpassung des Tests wird das LLM mit dem Prompt, der alle Informationen über die aktuell verfügbaren Testschritte, Karteninformationen und Testerstellungsregeln beinhaltet, ausgestattet. Nachdem der Test erstellt oder angepasst wurde, soll die Validierung beginnen. Somit soll erreicht werden, dass jede Antwort des Gherkin-Agenten vollständig ist und der Gherkin-Test bereits überprüft wurde. Nach der Validierung des Gherkin-Tests durch das Validierungs-LLM wird der Gherkin-Test an den externen Service zur Cucumber-Validierung gesendet und in die finale Antwort des Agenten eingebunden.

Alle Antworten der Agenten werden gesammelt und in einem finalen Schritt zusammengefasst. Diese Antwort wird an den Benutzer weitergeleitet und in der Benutzeroberfläche angezeigt. Die ausführliche Struktur ist in Abbildung 2 zu sehen.

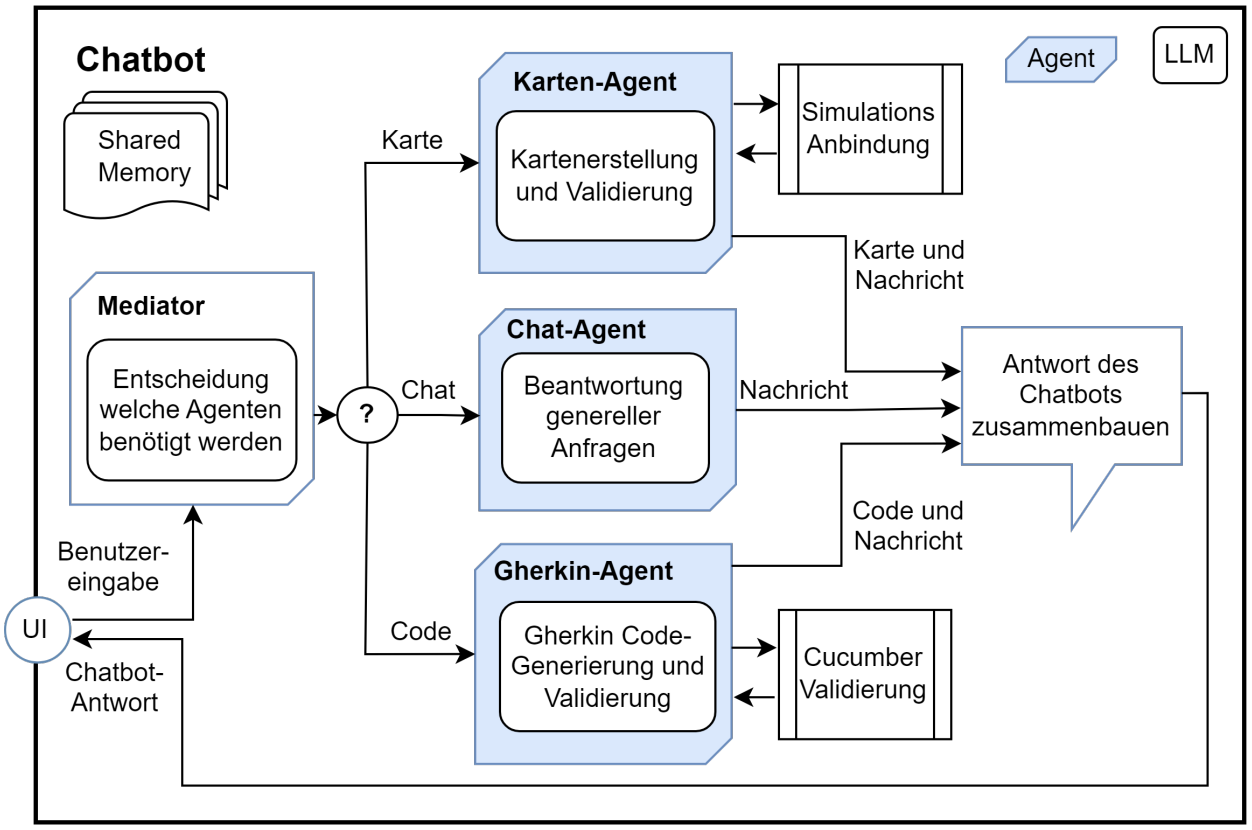

Abbildung 2: Veranschaulichung der Struktur des Chatbots mit Agenten-Ansatz. Dargestellt ist der Mediator sowie die drei Agenten und der Informationsfluss.

Quelle: Eigene Abbildung.

4 Related Work

BDD und die (teil-)automatisierte Erstellung von Testszenarien mit Gherkin und Cucumber beschäftigen unterschiedliche Bereiche. Die Ansätze zur Erstellung dieser Tests reichen von datenbankbasierten Methoden bis hin zu dem in dieser Arbeit aufgegriffenen Chatbot-Ansatz.

Beispielsweise beschäftigte sich Anzhan [30] in seiner Masterarbeit mit der automatisierten Generierung und Ausführung von Cucumber-Szenarien auf Basis eines relationalen Datenbankschemas. Im Gegensatz zur vorliegenden Arbeit wurde ein relationales Datenbankmanagementsystem (RDBMS) zur Modellierung der Beziehungen und Einschränkungen des Softwaresystems in einem Schema eingesetzt. Dadurch sollten frühzeitig User-Storys generiert werden, um Gespräche zwischen Entwicklern und Stakeholdern über Anwendungskonzepte und deren Beziehungen zu ermöglichen, ohne auf technische Datenbankbegriffe wie Felder, Datentypen und Einschränkungen zurückzugreifen.

Pena [31] beschreibt in seiner Masterarbeit aus dem Jahr 2018 den Einsatz von KI im Softwareentwicklungsprozess und erwähnt die Vorteile des Schreibens der Anforderungen in Gherkin zur Testerstellung. Zugleich beschreibt er die Herausforderungen, die sich durch den textuellen Ansatz in Bezug auf Unübersichtlichkeit und Komplexität ergeben. Er folgt dem Ansatz, einen Chatbot namens Specbot zu entwickeln, der aus in Gherkin beschriebenen Anforderungen Spezifikationen in einer weiteren Sprache, RSpec, erstellt. Hierdurch soll das manuelle Übersetzen von Gherkin in Code übernommen werden. Dieser Prozess wird in der vorliegenden Arbeit von dem erwähnten Tool Cucumber übernommen.

Winkler [32] untersucht die Entwicklung eines Chatbots mithilfe von GPT, um Cucumber-Szenarien zu generieren und damit End-to-End-Tests zu verbessern. Er beschreibt die Herausforderungen von End-to-End-Tests und wie KI-gestützte Tools helfen können, diese zu minimieren. Zugleich geht er auf die Problematik von Cucumber ein, bei dem bereits kleine Syntaxfehler dazu führen können, dass der gesamte Test fehlschlägt. Winkler hofft, durch die Integration eines Chatbots mit einem speziell hierfür angepassten Modell zur Generierung von Gherkin-Phrasen die genannten Probleme und Herausforderungen zu minimieren.

Mit einem anderen Fokus beschäftigte sich Kwame Adu [33] in seiner Masterarbeit mit der Testerstellung durch GPT. Hierbei betrachtete er Prompt Engineering, Finetuning und Retrieval-Augmented-Generation (RAG)[1]. Er betont, wie auch Giray [10] die Wichtigkeit,

[1] RAG ist eine Technik, die die Genauigkeit und Zuverlässigkeit durch das Abrufen von Informationen aus externen Quellen von GenAI-Modellen steigern soll [33, S. 20].

beim Definieren der Prompts eine Balance zwischen Verständnis und Prägnanz zu finden, um Verwirrungen des Modells zu verhindern. Beim Finetuning, welches das Erweitern des Modells mit spezifischen Daten beschreibt, konnte er eine anschließend erhöhte Generierung relevanter Szenarien beobachten. Gleichzeitig beschreibt er die dafür erforderliche hohe Menge an schwer zu erstellenden, hochwertigen und spezifischen Daten sowie den damit verbundenen Aufwand, ein Mittelmaß zur Vermeidung von Overfitting[1] zu finden. Der RAG-Ansatz unterstützte die Arbeit, da es dem Modell erlaubte, auf weitere Daten zuzugreifen.

Karpurapu et al. [34] untersuchten die Automatisierung von BDD-Akzeptanztests, indem unterschiedliche LLMs wie GPT-3.5, GPT-4 und Llama-2-13B verwendet und zwei Prompttechniken, Zero-Shot und Few-Shot, eingesetzt wurden. Beim Zero-Shot-Prompting werden, anders als beim Few-Shot-Prompting, keine vollständigen Beispiele für gewünschte Ein- und Ausgaben beigefügt. Die beiden GPT-Modelle GPT-3.5 und GPT-4 zeigten eine bessere Performance beim Erstellen von fehlerfreien BDD-Akzeptanztests. Zusätzlich lieferte die zweite Prompttechnik, Few-Shot-Prompting, höhere Genauigkeit durch das Einbeziehen von Beispielen. Zur Messung der Genauigkeit wurde ein Linting-Tool[2] verwendet und anhand von Kategorien wie dem Fehlen bestimmter Keywords zwischen den Modellen und Prompttechniken verglichen.

Shi et al. [36] beschäftigten sich mit der Wirksamkeit von ChatGPT beim Verfeinern von Gherkin-Spezifikationen. Nachdem Spezifikationen mit und ohne ChatGPT-3.5 verfeinert wurden, befragten sie Stakeholder nach ihrer Meinung zu den unterschiedlichen Verfeinerungen. Hierbei konnte kein signifikanter Unterschied zwischen der Zufriedenheit mit den verfeinerten Spezifikationen festgestellt werden. Die Studie zeigte jedoch, dass ChatGPT detaillierte Spezifikationen generierte, dabei aber Fehler in der Formulierung machte. Es wird empfohlen, aus den Antworten zu lernen und den KI-Chatbot entsprechend weiter zu verbessern. Das Verfeinern beinhaltete die Nutzung von Stakeholder-Kommentaren und Vision-Videos, um die Gherkin-Spezifikationen zu präzisieren und zu konkretisieren.

Die automatisierte Ausführung von Gherkin-Test-Spezifikationen mithilfe von LLM-Agenten wurde von Bergsmann et al. [27] untersucht und beinhaltete die Nutzung des AutoGen-Multi-Agenten-Frameworks[3]. Das System untersuchte selbstständig das System und war in der

---

1 Durch Overfitting ist das Modell zu stark auf die Trainingsdaten angepasst und verliert die Fähigkeiten neues zu generieren [10, S. 2632],[33, S 95].

2 Das von Karpurapu et al. verwendete Linting-Tool ist auf GitHub zu finden [35].

3 Weitere Informationen zu AutoGen sind unter https://microsoft.github.io/autogen/0.2/docs/Use-Cases/agent_chat nachzulesen.

Lage, aus den Testszenarien ausführbaren Testcode zu generieren und zu bewerten. Auch wenn hohe Erfolgsraten in einfachen und komplexeren Szenarien beobachtet wurden, traten Herausforderungen bei der Fehlererkennung auf. In Experimenten mit 15 Gherkin-Szenarien auf einer Demo-Anwendung zeigten LLMs vielversprechende Ergebnisse, selbst in komplexen Szenarien mit mehreren Schritten. Bergsmann et al. wiesen jedoch auf Probleme wie die Halluzination von Erfolgen und die Notwendigkeit präziser Spezifikationen und effektiven Prompt-Engineerings hin [27, S. 12ff.].

Relevant für diese Arbeit ist auch die bereits im Jahr 2012 von Soeken et al. [22] am Institut für Informatik der Universität Bremen beschriebene teil-automatisierte Methode, um mithilfe von NLP Szenarien in natürlicher Sprache in Quellcode umzuwandeln. Hierbei tritt der Benutzer in einen Dialog mit dem Computer, der Quellcode generiert und den Benutzer durch den Prozess führt. Der Benutzer kann diesen Code akzeptieren oder ablehnen, um so schrittweise aus den beschriebenen Szenarien Quellcode zu erstellen. Dies stellte den ersten Schritt zur Automatisierung von BDD dar und brachte erste Vorteile in Bezug auf die Effizienz von BDD und die Minimierung von Missverständnissen.

Die Vorgehensweise in der Arbeit von Soeken et al. umfasst vier Schritte: Das Szenario wird beschrieben, die Schrittdefinition wird erstellt, ein Code-Skelett wird generiert und abschließend werden die Operationen im Code-Skelett implementiert. Unterstützt wird dies durch die Erstellung von UML-Klassen- und Sequenzdiagrammen.

Aufgrund des Fokus der Bachelorarbeit wird auf den zweiten Schritt, die Erstellung der Schrittdefinition, kurz eingegangen. Erst wird die natürliche Sprache analysiert, um die relevanten Informationen zu extrahieren. Als Nächstes werden reguläre Ausdrücke erstellt, die die Struktur des Satzes erfassen und abschließend ein Codeblock erstellt, der ausgeführt wird, wenn dieser mit dem regulären Ausdruck übereinstimmt. Diese schrittweise Vorgehensweise ermöglicht das Überführen von Szenarien in natürlicher Sprache in ausführbaren Code. Die Verwendung regulärer Ausdrücke dient hier der Beschleunigung und Reduzierung der Fehleranfälligkeit. Zudem dient die interaktive Rückmeldung der kontinuierlichen Verbesserung und Anpassung der generierten Schrittdefinitionen und Code-Skelette.

Insgesamt kann zusammengefasst werden, dass das Thema unterschiedliche Facetten und Möglichkeiten zur weitergehenden Forschung besitzt. Im Gegensatz zu einigen der vorgestellten Arbeiten liegt der Fokus in der vorliegenden Arbeit auf einem Chatbot mit modernen GenAI-Modellen, der durch weitere Fähigkeiten wie das Integrieren und Erstellen

einer virtuellen Karte parallel zum Testerstellungsprozess ergänzt wird. Aus der Literatur geht hervor, dass GPT-Modelle bessere Ergebnisse als andere lieferten und ein Dialog zur kontinuierlichen Verbesserung weitere Vorteile mit sich bringt. Teilweise wurden diese Modelle speziell für den Anwendungsfall angepasst, worauf jedoch in dieser Arbeit verzichtet wird, um die Austauschbarkeit des Modells nicht zu verlieren. Die Anpassungen sowie benötigten Informationen werden über den Prompt übergeben. Nichtsdestotrotz werden Anregungen in Bezug auf die Erstellung des Prompts, das Verwenden eines Agentenansatzes sowie die Empfehlung, aus den Antworten zu lernen, durch das Speichern des Chatverlaufs berücksichtigt.

5 Implementierung

Dieses Kapitel befasst sich mit der Auswahl der Technologien und den während der Arbeit aufgetretenen Herausforderungen sowie den dafür implementierten Lösungen.

5.1 Auswahl der Technologien

Im Folgenden werden die in der vorliegenden Arbeit verwendeten Technologien und Werkzeuge beschrieben. Zunächst werden die Technologien für das Frontend beschrieben, gefolgt von den Technologien für das Backend und abschließend die Technologien für den Überprüfungsservice.

5.1.1 Frontend-Technologien

Für das Frontend wurde React[1] gewählt, da es im Vergleich zu Angular[2], einem weiteren modernen Frontend-Framework, eine einfachere Einarbeitung und höhere Geschwindigkeit bietet. Zudem eignet sich React besonders gut für reaktive Seiten. React ist eine auf JavaScript basierende Bibliothek zur Entwicklung von Benutzeroberflächen. Um mit React zu arbeiten, sind Grundkenntnisse in HTML, CSS und JavaScript erforderlich. HTML beschreibt die Struktur der Benutzeroberfläche, CSS die Darstellung, und mit JavaScript wird die Oberfläche durch Interaktionsmöglichkeiten erweitert. Das Zusammenspiel dieser drei Komponenten bildet die Grundlage für React [37, S. XVff.].

Wichtig hierbei ist jedoch, dass sich die Verwendung von JavaScript in React von der reinen Verwendung von JavaScript unterscheidet. Während bei der reinen Verwendung über den sogenannten Document Object Model (DOM)-Baum mithilfe von Methoden, wie dem Suchen nach einer bestimmten ID, der Inhalt der Elemente aktualisiert oder geändert wird, setzt React auf ein Konzept mit einem virtuellen DOM. Bei React werden zwei Kopien des DOM erstellt und im Speicher abgelegt. Dies ermöglicht es, nach dem Ändern einer der beiden Kopien diese mit der unveränderten Kopie zu vergleichen. Dieser Prozess, auch Diffing genannt, ermöglicht das Bündeln von Änderungen, die dann auf das reale DOM angewendet werden. Dadurch wird das Neu-Laden der Seite minimiert und die Leistung erhöht [37, S. 7ff., 13ff.].

[1] Weitere Informationen zu React sind in der Literatur nachzulesen [37].
[2] Weitere Informationen zu Angular sind in der Literatur nachzulesen [38].

5.1.2 Backend-Technologien

Für das Backend wurde die Programmiersprache Python gewählt, da bereits grundlegende Erfahrungen in Python vorhanden waren und die Sprache gut geeignet ist für Machine Learning und GenAI Projekte. Um die Abhängigkeiten von externen Bibliotheken zu organisieren, wurde das im Unternehmen weit verbreitete Dependency-Management-System Poetry[1] verwendet. Poetry ermöglicht eine übersichtliche Konfiguration der verwendeten Bibliotheken und deren Versionen.

Für die Validierung und Erzeugung von Objekten wurde Pydantic verwendet. Pydantic erlaubt es, unstrukturierte Daten in Objekten abzubilden und bietet eine Datenvalidierung und Datenkonvertierung zur Sicherstellung der Datenintegrität. Diese Objekte sind leicht zu erzeugen und zu strukturieren. Zudem ist ein Verschachteln dieser Objekte möglich, um komplexere JSON-Schemas abzubilden. In Listing 3 ist ein einfaches Pydantic-Objekt dargestellt.

```
1 class Test(BaseModel):
2     name: str
3     content: str
4     status: bool
```

Listing 3: Beispiel einer Pydantic-Klasse die einen Test darstellt. Man sieht die strukturierte Auflistung der Felder innerhalb der Klasse.

Quelle: Eigenes Listing.

Um eine Verbindung zum beschriebenen Frontend sowie der vAGV-Simulation herzustellen, ist eine Bibliothek zur Erstellung von APIs erforderlich. Hierfür kommt das Python-Framework FastAPI zum Einsatz. FastAPI bietet den Vorteil, gut mit Pydantic zu harmonieren und dessen Vorteile zu integrieren.

Als letzte hier genannte Bibliothek dient LangChain, um eine Verbindung mit dem LLM herzustellen. Da LangChain im Unternehmen wie Poetry weit verbreitet ist und ebenfalls Vorteile durch die Verwendung von Pydantic liefert, wurde LangChain in der vorliegenden Arbeit verwendet. LangChain[2] ist ein Framework, das bei der Entwicklung von Systemen mit LLMs dabei unterstützt, die Integration und Verwaltung von Sprachmodellen zu vereinfachen, indem es standardisierte Schnittstellen und Werkzeuge zur Verfügung stellt.

---

1 Zusätzliche Informationen zu Poetry sind auf der offiziellen Seite www.python-poetry.org nachzulesen.

2 Weitere Informationen zu LangChain sind auf der offiziellen Seite www.langchain.com nachzulesen

5.1.3 Validation Service Technologien

Für die bereits erwähnte externe Validierung mithilfe von Cucumber zum Parsen der einzelnen generierten Gherkin-Schritte werden eine andere Programmiersprache sowie andere Bibliotheken benötigt. Da die vAGV-Simulation, welche mit Cucumber arbeitet, in der Programmiersprache Kotlin geschrieben wurde und das Cucumber-Framework mit dieser Sprache kompatibel ist, wurde für den externen Validierungsservice ebenfalls Kotlin gewählt. Zudem sind hierbei die in Abschnitt 2.3 genannten Schritt-Definitionen Bestandteil des Services. Für die Verbindung mit dem Backend dient hier ktor[1], eine Bibliothek für das Erstellen von Client-Server-Anwendungen.

5.2 Auswahl der Chatbot-Architektur

Die in Abschnitt 3.2 beschriebenen zwei Ansätze zum Aufbau des Chatbots wurden beide bis zu einer funktionsfähigen Version implementiert und anschließend untersucht. Erste Tests zeigten, dass die Antworten, die im ersten Ansatz generiert wurden, teils ungenau waren. Hierbei fehlten Informationen wie Vorschläge, Rückfragen in den Antworten oder die für die Kartenerstellung und Gherkin-Testerstellung hinterlegten Regeln wurden nicht berücksichtigt. Dahingegen brachte der zweite Ansatz mit den spezialisierten Agenten in den ersten Tests bessere Ergebnisse hinsichtlich der genannten Ungenauigkeiten. Zudem wurden die Regeln für die Karten- und Gherkin-Testerstellung hierbei besser berücksichtigt.

In Verbindung mit den genannten Problemen und der durch den zweiten Ansatz verbesserten Codestruktur hinsichtlich der Austauschbarkeit einzelner Komponenten, wurde der zweite Ansatz für die vorliegende Arbeit gewählt. Da die weitere Differenzierung und vollständige Untersuchung der Vorteile einzelner spezialisierter System-Prompts gegenüber einem großen System-Prompt nicht Teil dieser Arbeit ist, wird darauf im Folgenden nicht weiter eingegangen.

Für die Implementierung des ausgewählten Chatbot-Designs wurde das Mediator-Pattern gewählt. Das Mediator-Pattern sieht vor, dass der Mediator zwischen den sogenannten Colleague-Klassen kommuniziert und vermittelt. Hier erfolgt jegliche Kommunikation über den Mediator, und es gibt keine direkte Kommunikation zwischen den Colleague-Klassen, was eine lose Kopplung unterstützt [39, S. 338ff.]. In der vorliegenden Arbeit wird dieser Ansatz leicht abgeändert, beispielsweise indem die genannten Colleague-Klassen ihren Mediator nicht kennen, sondern lediglich von ihm verwendet werden. Die Experten-Agenten dienen als

[1] Weitere Informationen zu ktor sind unter www.ktor.io nachzulesen.

Colleague-Klasse wobei alle Antworten der Experten gesammelt und wenn nötig zu einer finalen Antwort zusammengefasst werden. Hierdurch können kombinierte Fragen gesendet werden, die durch den Mediator in einzelne Aufgabenpakete unterteilt und verteilt werden.

5.3 Herausforderungen

Dieser Abschnitt beschäftigt sich mit den im Kapitel 3 genannten Anforderungen, der Architektur und den damit verbundenen Herausforderungen sowie den implementierten Lösungen. Dabei wird ein Einblick in den Code gegeben und die Umsetzung ohne tiefgehende technische Details beschrieben.

5.3.1 Verbindung zwischen Frontend und Backend

Für das Herstellen einer Verbindung zwischen Frontend und Backend war es notwendig, im Backend eine Anwendung zu erstellen. Hierfür wurde das vorgestellte Framework FastAPI verwendet. Nach dem Erstellen der Anwendung mussten die benötigten Endpunkte definiert werden. Insgesamt wurden fünf Endpunkte erstellt.

Bei dem wichtigsten Endpunkt, dem Chat-Endpunkt, wird eine Nachricht an die Anwendung gesendet und nach der Bearbeitung dieser Nachricht im Backend wird eine neue Nachricht zurück an das Frontend gesendet. Dabei kommt auch der genannte Vorteil von FastAPI zum Einsatz, gut mit Pydantic zu funktionieren. Die Eingangsnachricht sowie die Ausgangsnachricht sind in beiden Fällen Pydantic-Objekte. Nachdem der Endpunkt definiert wurde, muss die Funktion geschrieben werden, die nach dem Aufrufen des Endpunktes ausgeführt werden soll. In diesem Fall wird die Funktion *chat* definiert, welche den Chatbot aufruft, um eine Antwort auf die Eingangsnachricht zu generieren. Anschließend wird überprüft, ob die Antwort einen Gherkin-Test enthält und dieser entsprechend vorbereitet werden muss, sowie ob die Nachricht eine Empfehlung für den nächsten Schritt enthält. Alle Informationen werden in dem Pydantic-Objekt gemeinsam zurückgesendet.

Im Frontend muss anschließend an den im Backend erstellten Endpunkt eine POST-Anfrage gesendet werden. Im Body dieser Anfrage wird die Nachricht des Benutzers mitgegeben in Form eines JSON-Strings.

5.3.2 Auswahl der Agenten mithilfe des Mediators

Durch die Verwendung des Agenten-Ansatzes muss, wie in Abbildung 2 gezeigt, der Mediator eine Auswahl zwischen einem oder mehreren Agenten treffen. Hierbei wird in die dargestellten drei Kategorien Gherkin, Karte oder Allgemein unterteilt. Dieser Prozess wird durch ein LLM übernommen, welches die Auswahl anhand von Beispielen und definierten Auswahlkriterien im System-Prompt bestimmt. Zudem wird der Chatverlauf dem Prompt beigefügt, da die Antworten des Benutzers ohne Berücksichtigung der vorhergegangenen Antworten und Fragen nicht eindeutig eingeordnet werden können.

Um die Auswahl möglichst einfach zu gestalten, wurde das Pydantic-Objekt, welches der Mediator generiert, so strukturiert, dass dieses eine Liste eines weiteren Pydantic-Objekts enthält, welches jeweils das gewählte Thema und die auszuführende Anweisung enthält. Das Thema selbst besteht aus einem Enum-Objekt, wodurch lediglich eine der drei Varianten - gherkin, chat oder general - für das Thema ausgewählt werden kann.

Der Prompt des Mediators enthält im Wesentlichen sieben Abschnitte, wobei zunächst die Rolle und Aufgabe beschrieben wird. Anschließend werden drei Sonderfälle beschrieben, bei denen es um das Ersuchen von Hilfe durch den Benutzer geht. In diesem Fall ist dies zunächst ein allgemeines Anliegen, das sich jedoch durch den Kontext und die aktuelle Aufgabe, wie das Erstellen eines Gherkin-Tests, auf ein spezifisches Problem bezieht, und somit der dafür zuständige Agent antworten muss (siehe Zeile 8-11 in Listing 8). Nachdem die Sonderfälle definiert wurden, werden einige allgemeine Regeln definiert (siehe Zeile 13-20 in Listing 8) und daraufhin werden die vorgestellten Agenten, hier Experten genannt und vorgestellt (siehe Zeile 22-31 in Listing 8) mit den Regeln, in welchen Fällen der jeweilige Agent ausgewählt werden sollte (siehe Zeile 37-51 in Listing 8). Abschließend werden die Regeln für die Antworten und einige Beispiele in dem Prompt integriert.

Die Antwort des Mediators wird anschließend innerhalb der Chatbot-Klasse interpretiert und die Agenten durch eine IF-ELSE-Bedingung aufgerufen. Eine Besonderheit, wenn der Gherkin-Agent ausgewählt wurde, ist, dass eine erneute Prüfung innerhalb der Chatbot-Klasse erforderlich ist. Die vom Gherkin-Agenten generierte Antwort enthält ein Karten-Objekt, welches im Rahmen der Testerstellung neue Informationen zu den platzierenden AGVs und den aufzunehmenden Ladungen liefern kann. Die Prüfung stellt sicher, dass bei einer Änderung der Informationen der Karten-Agent diese in der angezeigten Karte anpasst.

5.3.3 Genereller Chat Agent

Der generelle Chat-Agent übernimmt die Aufgabe, mithilfe eines LLMs generelle Fragen und Fragen über das System zu beantworten. Der Prompt des LLMs enthält zunächst ebenfalls eine Aufgabe, generelle Regeln sowie spezifische Regeln über das Antwortformat. Anschließend wird in zwei abgegrenzten Abschnitten auf die Funktionen des Systems und die Komponenten eingegangen. Diese werden ausführlich beschrieben, um dem Benutzer möglichst detaillierte Angaben liefern zu können (siehe Zeile 18-38 in Listing 9). Abschließend enthält der Prompt ein Beispiel für eine vollständige Antwort.

5.3.4 Testerstellung und Validierung innerhalb des Gherkin-Agenten

Der Gherkin-Agent ist, wie in Abschnitt 3.2 beschrieben, zuständig für das Erstellen, Anpassen und Validieren von Gherkin-Tests. Im Kern besteht der Gherkin-Agent aus drei internen Komponenten und folgt einem klar definierten Prozess. Zunächst wird mit einem LLM die Benutzeranfrage überprüft und entschieden, ob der Gherkin-Test angepasst werden muss oder ob es sich um die Beantwortung einer Frage zu bestehenden Gherkin-Testabschnitten handelt. Diese Vorentscheidung trägt dazu bei, unnötige LLM-Aufrufe zu verhindern, die durch das Validieren des Gherkin-Tests entstehen, auch wenn dieser nicht angepasst wurde. Handelt es sich bei der Benutzeranfrage um eine generelle Frage, bei der kein Gherkin-Test angepasst werden muss, wird die Antwort auf diese Frage durch dasselbe LLM generiert, das die Entscheidung trifft, ob der Gherkin-Test generiert oder geändert werden muss. Andernfalls wird die Anfrage an die nächste Komponente weitergeleitet, die alle Informationen bezüglich der Regeln für die Gherkin-Testerstellung und -anpassung kennt.

Nachdem der Gherkin-Test erstellt oder angepasst wurde, beginnt die Validierung durch ein LLM, gefolgt von der Struktur-Validierung durch den externen Service, auf welchen in einem getrennten Abschnitt genauer eingegangen wird. Nach erfolgreicher Validierung wird die Antwort des Gherkin-Agenten zusammengebaut und auf potenziell geänderte platzierbare Objekte wie AGVs überprüft und gegebenenfalls in dem dafür vorgesehenen Objekt angepasst. Der Prozess innerhalb des Agenten ist in Abbildung 3 nochmals dargestellt.

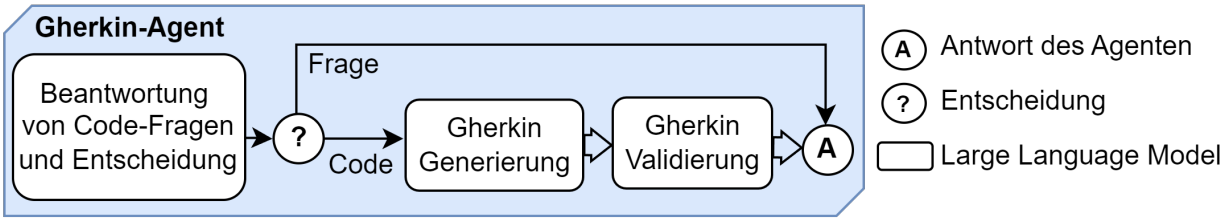

Abbildung 3: Prozess innerhalb des Gherkin-Agenten. Zu sehen sind die drei Hauptkomponenten des Gherkin-Agenten.

Quelle: Eigene Abbildung.

Die wichtigsten zwei Prompts hierbei sind der Prompt zur Generierung und Validierung des Gherkin-Tests. Bei der Erstellung oder Anpassung des Gherkin-Tests wird dem Prompt unter anderem der existierende Test, die existierende Karte und die verfügbaren Gherkin-Schritte übergeben. Um eine Anpassung der verfügbaren Gherkin-Schritte zu ermöglichen, ohne den Prompt selbst anpassen zu müssen, werden diese über eine externe JSON-Datei eingelesen. Die Besonderheit der in dieser Datei enthaltenen Gherkin-Schritte ist ihre Zusammensetzung. Neben dem einzelnen Gherkin-Schritt werden hier Vor- und Nachbedingungen definiert, also Schritte, die unmittelbar vor oder nach dem beschriebenen Schritt erfolgen müssen. Dies wird zudem in einer kurzen Zusammenfassung beschrieben (siehe Listing 4).

```
{
    "step": "When the vagv <agv_id: str>  drives to node <node_id: int>",
    "additional_pre_conditions_steps": [
        "Given the vagv <agv_id: str> waits on node <node_id: int>"
    ],
    "post_conditions_steps": [
        "Then the vagv <agv_id: str> is at node <node_id: int>"
    ],
    "condition_description": "Before agv can drive, agv must be defined.
        After agv drives to node, status must be checked."
}
```

Listing 4: Beispiel eines Gherkin schrittes mit Vor- und Nachbedingungen. Zu sehen ist der Schritt der ein AGV zu einem Knoten fahren lässt.

Quelle: Eigenes Listing.

Der Prompt beinhaltet, wie die Prompts des Mediators und des generellen Chat-Agenten zunächst die Rolle und Aufgabe, gefolgt von einigen generellen Regeln. Danach beginnt der spezifische Abschnitt für die Testerstellung, welcher in vier Guidelines unterteilt worden ist. In

der ersten Guideline werden die initialen Schritte aufgelistet. Diese umfassen im ersten Schritt das Fragen nach dem Feature-Namen gefolgt von der Frage nach dem Szenario-Namen. Diese Informationen sind die Voraussetzungen einen Test erstellen zu können und müssen somit als Erstes erfolgen. Der dritte initiale Schritt ist das Hinzufügen erster Schritte, die bei jedem Test enthalten sein müssen. Durch das automatisierte Hinzufügen dieser Schritte muss der Benutzer die Anforderung nicht kennen, sondern kann direkt mit dem Erstellen des gewünschten Tests starten (siehe Zeilen 9-15 in Listing 10).

Eine weitere Anforderung an das System ist es, dem Benutzer möglichst viel Schreibarbeit abzunehmen. Dies kann durch das Erstellen einer Grundstruktur des Tests nach der Eingabe des Szenario-Namens erfolgen. Der Szenario-Name beschreibt in der Regel die zu testende Situation wie *"The AGV drives from Node 1 to Node 2"*. In diesem Fall sollen die Schritte, die notwendig sind, um den Test zu erstellen, direkt integriert und der Benutzer darüber informiert werden. Da sich allerdings nicht darauf verlassen werden kann, dass der Szenario-Name ausreichend Informationen für die Testerstellung liefert, wird in der zweiten Guideline ein Vorgehen hierfür definiert (siehe Zeilen 17-26 in Listing 10).

Der Benutzer soll sowohl, wenn eine Grundstruktur erstellt wurde, als auch wenn keine erstellt wurde, die Möglichkeit haben, Änderungen am erstellten Test vorzunehmen. Hierbei sind einige Regeln zu beachten, wie mit den erwähnten Vor- und Nachbedingungen beim Hinzufügen, Ändern und Entfernen eines Schrittes umgegangen werden soll. Dieses Vorgehen, welches sich in die drei Fälle: hinzufügen, ändern und entfernen aufteilt, wird in der dritten Guideline beschrieben. Nach jedem beschriebenen Schritt werden zwei Optionen angeboten, um den Erfolgsfall und den Fehlerfall abzudecken und den nächsten Schritt wählen zu können. Hierdurch wird dem LLM eine klare Struktur mitgegeben (siehe Zeilen 28-54 in Listing 10).

Abschließend muss sichergestellt werden, dass der Gherkin-Test nun alle notwendigen Schritte mit Vor- und Nachbedingungen enthält. Das gewünschte Antwortformat des LLMs sowie eine abschließende Prüfung werden in der letzten Guideline nochmals beschrieben (siehe Zeilen 56-59 in Listing 10).

Zusätzlich werden einige spezifische Regeln zu den AGVs beschrieben, wie dass dieser nur auf vorhandene Routen fahren darf und jeweils nur ein AGV auf einem Knoten stehen kann, beschrieben. Abschließend enthält der Prompt einige Beispiele mit Benutzereingabe und einer möglichen Antwort.

Das zweite LLM, welches für die Testvalidierung zuständig ist, enthält ebenfalls einige Abschnitte im Prompt. Im ersten aufgabenspezifischen Abschnitt wird überprüft, ob die Änderungen am Gherkin-Test durch den Benutzer gewünscht wurden oder diese rückgängig gemacht werden müssen (siehe Zeilen 12-19 in Listing 11). Im nächsten und wichtigsten Abschnitt des Prompts werden Prüfkriterien anhand von Schritten definiert. Es definiert ein schrittweises Vorgehen mit einzelnen Kategorien. Die Kategorien sind:

a. **Initiale Regeln wurden eingehalten**: Hierbei wird beispielsweise geprüft, ob die im Gherkin-Test enthaltenen Schritte mit den, in der übergebenen JSON-Datei stehenden verfügbaren Schritten übereinstimmen.

b. **Initiale Schritte sind enthalten**: Jeder Test benötigt initiale Schritte, wie das Starten der Simulation und definieren der Karte. Dies muss überprüft und gegebenenfalls hinzugefügt werden.

c. **Feature und Szenario wurden benannt**: Um einen validen Gherkin-Test zu erhalten, müssen die Feature- und Szenario-Definition enthalten sein.

d. **Überprüfung der Vor- und Nachbedingungen**. Für jeden Schritt existieren mögliche Vor- und Nachbedingungen die geprüft und, wenn möglich, ergänzt werden müssen.

e. **AGV und Ladungen sind korrekt**: Die Regeln für die platzierbaren Objekte, darunter AGVs, müssen eingehalten werden um diese korrekt anzeigen zu können.

f. **Zusammenfassung und finale Prüfung**: Nachdem alle Kategorien durchgearbeitet wurden, wird eine abschließende Einschätzung vorgenommen.

Jede dieser Kategorien enthält eine oder mehrere sogenannte Checks, die einzelne, zu prüfende Kriterien enthalten. Zudem sind in jeder Kategorie zwei Optionen enthalten, die dem LLM bei der Wahl des weiteren Vorgehens helfen sollen. Eine der Optionen beschreibt immer den Erfolgsfall und die andere Option den Fehlerfall (siehe Zeilen 21-72 in Listing 11).

Dem Prompt wird in diesem Fall neben den Informationen, die bereits dem anderen LLM übergeben wurden, der neu erzeugte oder geänderte Gherkin-Test übergeben, wodurch die Änderungen ersichtlich werden und invalide Elemente entdeckt werden können. Sollten invalide Elemente entdeckt werden, wird dies mit den Gründen erneut dem LLM für die Testerstellung übergeben und eine erneute Prüfung des generierten Tests durchgeführt. Im Rahmen dieser Validierung wird auch der externe Cucumber-Validierungs-Service aufgerufen, auf den im nächsten Abschnitt eingegangen wird.

5.3.5 Cucumber Struktur Validierung

Wie im vorherigen Abschnitt erwähnt, wird ein externer Dienst zur Validierung der Struktur des Gherkin-Tests aufgerufen. Zunächst wird ein Singleton-Objekt definiert, das eine Methode enthält, die den übergebenen Gherkin-Test umwandelt, in einer Feature-Datei speichert und mithilfe des Cucumber-Comand Line Interface (CLI) ausführt. Der CLI wird ein Array von Argumenten übergeben. Die übergebenen Argumente umfassen den Pfad zur Datei mit dem übergebenen Gherkin-Test, das sogenannte Glue-Code-Paket, das die verfügbaren Gherkin-Schritte und Implementierungen enthält, und schließlich ein Plugin-Argument, um die Ergebnisse der CLI im JSON-Format in einer Datei abzulegen (siehe Listing 11).

Abschließend wird das Ergebnis gefiltert, um nur die relevanten Informationen an den Gherkin-Agenten zurückzuliefern. Das Ergebnis wird in ein JSON-Objekt geparst und anschließend die einzelnen Schritte extrahiert. Es wird hierbei nur der jeweilige Schritt und der Status, ob dieser gültig oder ungültig ist, im Rückgabe-Ergebnis gespeichert.

```
1  object CucumberRunner {
2    @Throws(IOException::class)
3    fun runFeatureFromString(featureContent: String): String {
4      // Temp-Datei mit "featureContent" und Ergebnis-Datei wird erstellt
5      {...}
6
7      val argv = arrayOf(tempFile.absolutePath,
8          "--glue", "org.example.steps",
9          "--plugin", "json:$resultFile"
10     )
11     CucumberMain.run(*argv)
12     // Ergebnis-Datei wird gelesen, geparst, gefiltert und zurueckgegeben
13     {...}
14   }
15 }
```

Listing 5: Ausschnitt aus dem Cucumber-Validierungs-Service. Zu sehen ist der Aufruf der Cucumber-CLI und das extrahieren der relevanten Informationen aus dem Ergebnis der CLI.

Quelle: Eigenes Listing.

Das Ergebnis dieser letzten Validierung entscheidet, wie in Abschnitt 3.1.3 erwähnt, über den Erfolg, einen gültigen Gherkin-Test erstellt zu haben. Dieses Ergebnis wird innerhalb des

Gherkin-Agenten interpretiert und entschieden, ob Anpassungen notwendig sind oder der Gherkin-Test in der Benutzeroberfläche angezeigt werden kann.

5.3.6 Karten Agent

Die dritte Hauptkomponente des Systems, der Karten-Agent basiert ebenfalls auf einem LLM, welches hier die Erstellung neuer Kanten oder Knoten übernimmt. Die Struktur der LIF-Karte wird durch ein Pydantic-Objekt abgebildet. Um dieses Objekt erstellen zu können, benötigt das LLM einen Prompt, welcher alle Informationen über das Kartenformat und die Regeln der Kartenerstellung enthält. Zudem wird auch dem Karten-Agenten der Chatverlauf an den Prompt angehängt, um diesen zu berücksichtigen.

Die wichtigsten zwei Abschnitte innerhalb des Promptes sind:

1. **Schritte und Vorgehen beim Kartenerstellen**: Um in den beiden vorgestellten Fällen, eine Karte wird geladen oder eine Karte wird erstellt entsprechend zu reagieren werden in diesem Abschnitt zwei Fälle beschrieben in denen die einzuhaltenden Schritte enthalten sind. Hierzu zählen welche Fragen in dem jeweiligen Fall gestellt werden müssen (siehe Zeilen 13-23 in Listing 12).
2. **Regeln für das Erstellen von Knoten und Kanten**: Das Erstellen der Knoten und Kanten stellt einen wesentlichen Aspekt der Kartenerstellung dar, um diesen Prozess korrekt auszuführen, benötigt es klare Regeln. Diese Regeln umfassen zum einen die Benennung der Knoten und Kanten sowie die Einschränkungen bei Platzierung (siehe Zeilen 25-33 in Listing 12).

Nachdem das LLM die Karte angepasst oder neu erstellt hat, wird eine Überprüfung durch ein weiteres LLM vorgenommen, ob lediglich die Änderungen vorgenommen wurden, die der Benutzer angefragt hatte. Hierbei wird verglichen, welche Änderungen im Karten-Objekt vorgenommen wurden. Diese Entscheidung wird durch Beispiele im Prompt unterstützt (siehe Listing 13). Abschließend werden vorhandene Änderungen direkt durch die Simulationsanbindung ausgeführt. Hierbei werden zunächst die Informationen verglichen und nur, wenn Änderungen festgestellt werden, die Knoten und Kanten angepasst. Anschließend werden die AGVs und Ladungen überprüft und entweder hinzugefügt oder entfernt. Des Weiteren ist eine Fehlerspeicherung notwendig, um die auftretenden Fehler in einem späteren Schritt dem LLM zurückzuspielen und Anpassungen vorzunehmen.

5.3.7 Simulationanbindung

Um die virtuelle Karte in der Simulation anpassen zu können, ist die Anbindung an die Simulation erforderlich. Hierbei existiert in der Simulation bereits eine Sammlung an Endpunkten für die Steuerung der Kartenelemente. Die Informationen können über HTTP-Requests an den jeweiligen Endpunkt gesendet werden. Die Simulationsanbindung teilt sich in zwei Klassen auf, um die Logik und das Aufrufen der Endpunkte voneinander zu trennen. Der Client übernimmt lediglich das Zusammenbauen der Anfragen und das Aufrufen der Endpunkte, wobei der Service auswählt, ob und welcher Endpunkt aufgerufen werden muss. Ein Beispiel im Service, welches das Hinzufügen eines AGVs zeigt, ist in Listing 6 zu sehen.

```
def create_agv(self, agv: AGV, map_information: MapInformation) ->
    SimulationResponse:
    if agv.id in map_information.map_placement_information.agvs:
        logger.debug("AGV already exists.")
        return SimulationResponse(
            success=False,  result_message="AGV already exists"
        )

    response = self.simulation_client.put(
        endpoint="/agvs", json_data=agv.model_dump_json()
    )

    # Prüfen ob response erfolgreich war und zurückgeben des Ergebisses
    {...}
```

Listing 6: Ausschnitt aus der Simulations-Anbindung. Zu sehen ist die Funktion, die einen AGV in der Karte hinzufügt und überprüft, ob die Anfrage ein Erfolg war.

Quelle: Eigenes Listing.

Da sich der Prozess, die Informationen in der Karte hinzuzufügen oder zu ändern, sehr stark ähnelt, wird auf die weiteren Varianten zum Anpassen der Karte über die Simulationsanbindung nicht weiter eingegangen.

6 Validierung

Um eine geeignete Evaluationsmethode zu finden wurde zusätzliche Literatur zur Evaluation von Chatbots untersucht, welche Ideen und Anregungen für eine finale Evaluation liefern soll. Diese wird im Folgenden zusammengefasst.

Im Rahmen der 36. Internationalen wissenschaftlichen Konferenz zu Ökonomie und Sozialentwicklung wurden insgesamt 15 Frameworks, wie das PARADISE-Framework zur Bewertung von Chatbots untersucht. Es wurde ein neues umfassendes Framework entwickelt, das durch das Integrieren von fünf Perspektiven – Benutzererfahrung, Informationsabruf, Linguistik, Technologie und Geschäft – die Bewertung des Chatbots aus verschiedenen Blickwinkeln ermöglicht [40, S. 100ff.]. Hierbei sind die folgenden zwei Perspektiven für die vorliegende Arbeit interessant [40, S. 104ff.]:

- **Benutzererfahrungsperspektive:** Diese Perspektive ist wesentlich, um zu verstehen, wie das System wahrgenommen wird und wie die Leistung des Chatbots hinsichtlich Vollständigkeit und Schnelligkeit sowie die Zufriedenheit der Benutzer gemessen wird. Um dies zu erreichen, wird neben anderen Metriken, das Durchführen einer Benutzerumfrage zur Erfassung des Benutzererlebnisses und der Benutzbarkeit vorgeschlagen.
- **Informationsperspektive:** Hier wird der Chatbot hinsichtlich des Informationsbedarfs sowie der Genauigkeit, Zuverlässigkeit und Effizienz der bereitgestellten Informationen analysiert. Für diese Analyse werden quantitative Metriken wie die Erfolgsquote empfohlen.

Die drei weiteren Perspektiven – Linguistik, Technologie und Geschäft – sind aufgrund der aktuellen Phase der Testentwicklung, die lediglich die Machbarkeit sowie Möglichkeiten untersucht, nicht im Fokus.

Abeysinghe et al. untersuchten im Rahmen des ersten Workshops des American Institutes for Research in Arlington im Juli 2024 zur Bewertung von LLMs für Information Retrieval ebenfalls die Herausforderungen bei der Evaluierung von LLM-Anwendungen. Die Wichtigkeit, effektive Bewertungsmethoden zu finden, wird aufgrund der Verbreitung solcher Anwendungen hervorgehoben. Weiter ist das grundlegende Bewerten der Basis-LLMs für eine effiziente Nutzung der Fähigkeiten und das Minimieren der Risiken in der Anwendung ein wesentlicher Bestandteil. Abeysinghe et al. beschreiben, dass Chatbots über verschiedene Dimensionen bewertet werden können. Diese Dimensionen umfassen beispielsweise die Benutzerfreundlichkeit,

Reaktionszeit, sprachliche Wirksamkeit oder die Möglichkeit, die Anfragen der Anwender vollständig zu beantworten. Um dies jedoch durchzuführen, existiert derzeit keine gängige Methode oder Best-Practices, die robuste Bewertungen eines solchen Systems zulassen. Es wird beschrieben, dass Entwickler und Forscher aufgrund der fehlenden Metriken und geeigneten Messgrößen die Verantwortung zur Wahl geeigneter Bewertungsmethoden für ihr einzigartiges System tragen. Bei der Evaluation durch einen Menschen wird das Verwenden einer 5- oder 7-Punkt-Likert-Skala empfohlen [41, S. 1ff.].

Auch Banerjee et al. haben die Wichtigkeit und Notwendigkeit verlässlicher Methoden und Metriken zur Bewertung von LLM-basierten Chatbots untersucht. Es wird ein End-2-End Benchmarking Framework entwickelt um einen Chatbot hinsichtlich Performance und Fähigkeiten standardisiert zu bewerten. Dieses Framework wurde für Chatbots, bei denen das Beantworten von Fragen im Vordergrund steht, entwickelt und ist ebenfalls in der vorliegenden Arbeit nur begrenzt anzuwenden. Neben dem Framework wurden weitere relevante Metriken für die Bewertung eines Chatbots wie Relevanz, Vollständigkeit und Genauigkeit genannt. Allgemein sollten Chatbots genau, relevant und nützlich sein [42, S. 1ff.].

Da sich die vorgestellten Frameworks und Metriken nicht vollständig für die Evaluierung des Systems in der vorliegenden Arbeit eignen, ist das Entwickeln eines eigenen, speziell für diesen Anwendungsfall geeigneten Vorgehens für die Evaluierung erforderlich. Einige Qualitätsmerkmale eines Chatbots, wie die Vollständigkeit der Antwort und vor allem die Benutzerfreundlichkeit, wiederholen sich und werden aus diesem Grund für die Entwicklung eines eigenen Evaluierungsvorgehens ebenfalls mit einbezogen.

Die finale Evaluation umfasst zwei Schritte. Im ersten Schritt wird das System anhand einer Sammlung von Testfällen automatisiert über einen Datensatz getestet. Im zweiten Schritt werden Experteninterviews mit 5-10 Teilnehmern, bestehend aus technischen Experten und potenziellen Anwendern des Systems, durchgeführt, um unter anderem die Benutzerfreundlichkeit zu erfassen. In den nachfolgenden Abschnitten werden die zwei Schritte nochmals spezifischer erklärt und die Ergebnisse vorgestellt.

6.1 Test basierte Evaluation

Wie bereits erwähnt, wird im ersten Schritt der Evaluation eine auf einem Datensatz basierende, automatisierte Evaluation durchgeführt. Ein vollständiger Test beinhaltet zunächst alle grundlegenden Informationen wie aktuelle Karten-Informationen, Gherkin-Test-Informationen

und eine Chat-Historie. Diese Informationen werden über das JSON-Objekt des Tests eingelesen und beim Erstellen des Testsetups dem Chatbot übergeben. Zudem wird eine Nachricht des Benutzers aus dem JSON-Objekt eingelesen, um die nächste Aktion des Chatbots anzustoßen. Jeder Test prüft hierbei ein bestimmtes Verhalten oder eine bestimmte Funktion des Systems und umfasst wiederum einen oder mehrere sogenannte Checks. Ein einzelner Check beschreibt einen bestimmten Aspekt oder ein zu prüfendes Kriterium innerhalb des Tests und besteht aus einer Beschreibung und den gewünschten Ausgaben. Um die automatisierte Evaluation der Tests zu erleichtern, wurde zudem ein Mock-Service für den Simulation-Client implementiert, wodurch keine laufende Simulation für die Evaluation notwendig ist. Die Struktur eines Tests ist in Listing 7 nochmals als Pydantic-Objekt dargestellt.

```
1 class TestCase(BaseModel):
2   name: str = Field("The name of the test case.")
3   memory: List[ChatMessage] = Field("The memory of the chatbot.")
4   map_information: MapInformation = Field("The existing map information.")
5   test_information: GherkinTest = Field("The existing test information.")
6   message: str = Field("The message that the user sends to the chatbot.")
7   checks: List[Check] = Field("The checks that should be performed.")
```

Listing 7: Inhalte eines Testcase dargestellt als Pydantic-Objekt. Zu sehen sind alle Attribute und Objekte innerhalb eines Testcase mit einer Beschreibung des Feldes.

Quelle: Eigenes Listing.

Nachdem die einzelnen Tests eingelesen wurden, werden diese schrittweise ausgeführt. Zunächst wird die nächste Ausgabe des Chatbots mit der erwähnten Benutzer-Nachricht angestoßen. Sobald der Chatbot eine Antwort geliefert hat, wird nacheinander jeder im Test enthaltene Check ausgeführt. Die verfügbaren neun Checks sind:

1. **Karte stimmt überein**: Es wird hierbei geprüft, ob die in der Antwort des Chatbots enthaltene Karte mit der im Check hinterlegten Karte übereinstimmt. Durch diese Prüfung können zwei Dinge festgestellt werden: Erstens, ob die gewünschten Änderungen in der Karte erfolgreich umgesetzt wurden, und zweitens, ob die Karte nicht ohne das Einverständnis des Benutzers geändert wurde.

2. **Karte enthält Knoten**: Durch die Verwendung eines LLMs sind verschiedene Eingaben bei der Kartenerstellung möglich. Der Benutzer muss nicht zwangsläufig die genauen Positionen der Knoten angeben, um diese zu erzeugen. Hierbei können beispielsweise auch Formen, wie das Positionieren der Knoten in einer Linie oder einem Kreis,

angefordert werden. Der Chatbot setzt dabei die Positionen der einzelnen Knoten entsprechend der Form, wodurch sich die erste Kategorie, die Karte entspricht exakt dem gewünschten, nicht anwenden lässt. Es wird in diesem Check lediglich auf die Existenz der angeforderten Knoten ohne Berücksichtigung der Positionen geprüft.

3. **LLM-Antwortüberprüfung**: Da es sich um ein LLM handelt und die Möglichkeit besteht, neben den eigentlichen Aufgaben des Systems weitere Anfragen, wie das Erzählen eines Witzes oder das Beleidigen des Systems, durchzuführen, muss in diesem Fall beispielsweise geprüft werden, ob das System weiterhin freundlich antwortet oder ob das System nach dem Erzählen eines Witzes wieder auf das ursprüngliche Thema zurückführt. Dies wird durch ein weiteres LLM in diesem Check geprüft und das Prinzip LLM-as-a-Judge angewendet.
4. **Prüfung des Agenten**: Um die erwartete Antwort auf die Benutzer-Nachricht zu erhalten ist die Auswahl des richtigen Agenten erforderlich. Dieser Check prüft hierbei, ob der richtige Agent durch den Mediator gewählt wurde.
5. **Gherkin-Test enthält Feature-Namen**: Der Gherkin-Test teilt sich in verschiedene Elemente auf. Dieser Check prüft, ob der Feature-Name korrekt in der Antwort des Chatbots gesetzt wurde.
6. **Gherkin-Test enthält Szenario-Namen**: Wie im vorherigen Check wird ein einzelnes Element des Gherkin-Tests überprüft. Hierbei liegt der Fokus auf dem Szenario-Namen in der Antwort des Chatbots.
7. **Karte enthält platzierbare Objekte**: Die platzierbaren Objekte müssen zunächst auf ihre Existenz überprüft werden. Dabei wird, wie in Check 2 nur geprüft, ob die AGVs oder Ladungen enthalten sind und nicht ob sie korrekt platziert sind.
8. **Platzierbare Objekte stimmen überein**: Im achten Check wird der vorherige Check erweitert und die platzierbaren Objekte auf die korrekte Platzierung überprüft.
9. **Gherkin-Test enthält Schritte**: Der neunte Check prüft das Vorhandensein der Gherkin-Testschritte. Dies kann sowohl für das Überprüfen der initialen Schritte zum Simulationsstart und der Kartendefinition als auch für das konkrete Prüfen, ob die angefragten, generierten Schritte enthalten sind, verwendet werden.

Ein Test kann, wie erwähnt, einen oder mehrere dieser Checks verwenden, um eine Fähigkeit oder ein Verhalten des Chatbots zu prüfen. Ein Beispiel für einen Test ist das Hinzufügen

des Szenario-Namens durch den Benutzer. Mit der Benutzereingabe *„I want to name my scenario AGV drives from node 1 to node 2“* sollten nach der Antwort des Chatbots folgende Anforderungen erfüllt sein:

- Die LIF-Karte sollte nicht angepasst worden sein.
- Der Szenario-Name sollte richtig gesetzt sein.
- Es sollten die initialen Gherkin-Schritte enthalten sein:
  1. Starten der Simulation („*Given the simulation is started*“).
  2. Definieren der Karte („*Given the lif map <map_name> defines the world*“).
- Durch den Szenario-Namen sollten bereits die drei folgenden Schritte enthalten sein:
  1. Platzieren des AGVs auf Knoten eins („*Given the vagv agv_1 waits on node 1*“).
  2. Starten der Fahrsequenz („*When the vagv agv_1 drives to node 2*“).
  3. Prüfen, ob der AGV angekommen ist („*Then the vagv agv_1 is at node 2*“).
- Es sollte in neuer AGV in der Karte für platzierbare Objekte enthalten und durch die Anweisung, auf Knoten zwei zu fahren, auf Knoten zwei platziert sein.

Für den genannten Test werden die Checks 1, 4, 6, 7, 8 und 9 benötigt, um das System vollständig zu testen. Die Ergebnisse der Checks sind in Tabelle 2 dargestellt und enthalten neben einer Beschreibung auch den Status, der zeigt, ob der Check erfolgreich war.

Used check	**Specific check description**	**Status**
1	Lif map should not be changed.	PASSED
4	The Gherkin-Agent should respond to the user.	PASSED
6	Scenario name is set to AGV drives from node 1 to node 2.	PASSED
7	Placement map should include a new agv.	PASSED
8	Placement map should include a new agv at the correct node 2.	PASSED
9	Gherkin-Test includes initial steps.	PASSED
9	Gherkin-Test includes steps based on scenario description.	PASSED

Tabelle 2: Beispiel eines Testergebnisses mit mehreren Checks nach der Benutzereingabe *„I want to name my scenario AGV drives from node 1 to node 2“*. Zu sehen sind sieben Checks, die alle bestanden wurden.

Quelle: Eigene Tabelle.

Analog zum gezeigten Beispiel können durch das Kombinieren der vorgestellten Checks unterschiedliche Tests durchgeführt und die Antworten des Chatbots überprüft werden. Somit

kann auf Basis des Datensatzes, welcher mehrere Tests enthält, automatisiert geprüft werden, ob die einzelnen Tests erfolgreich waren, und wenn nicht, anhand der einzelnen Checks mögliche Fehler identifiziert werden. Zudem können einzelne Checks durch die Anpassung ihrer Parameter, wie im Beispiel die enthaltenen Schritte bei Check 9, mehrfach in einem Test verwendet werden.

Die Tests können in drei Kategorien, basierend auf den drei Agenten, eingeteilt werden:

- **Allgemein:** Diese Tests beziehen sich auf den Chat-Agenten und umfassen das Beleidigen des Systems oder die Prüfung, inwieweit das System nach einem allgemeinen, für die Karten- oder Testerstellung nicht relevanten Thema zurückführt.
- **Kartenerstellung:** Bei den Tests für den Karten-Agenten werden insbesondere da Hinzufügen und Ändern von Knoten und Kanten durch direkte und indirekte Benutzereingaben überprüft. Direkte Angaben sind hierbei die Positionen oder Start- und Endknoten wobei indirekte Angaben lediglich die Anweisung einen oder mehrere Knoten in einer bestimmten Relation oder Form zu platzieren und Kanten beispielsweise durch die Anweisung, die Knoten im Uhrzeigersinn zu verbinden beinhalten.
- **Testerstellung:** Die letzte Kategorie der Tests beschäftigt sich mit dem Testen des Gherkin-Agenten. Wie in dem vorgestellte Beispiel, das die Eingabe eines Szenario-Namen prüft, muss der Chatbot die Anweisungen befolgen und auf Basis des Szenarios bereits grundlegende Schritte erzeugen. Des Weiteren umfassen diese Tests das Ändern und das Hinzufügen von Testschritten vor und nach den grundlegenden Schritten. Weiter wird das Erklären eines Fehlers, wenn der Benutzer einen unmöglichen Schritt, wie das Platzieren eines AGVs auf einem nicht existierenden Knoten, hinzufügen möchte und das Löschen einzelner Schritte, woraufhin die davon abhängigen Schritte entfernt werden müssen getestet.

Die vollständige Liste von Tests unterteilt sich in 6 Generelle-Tests mit 13 Checks, 15 Gherkin-Tests mit 64 Checks und 18 Karten-Tests mit 39 Checks. Insgesamt ergeben sich hieraus 39 Tests mit insgesamt 116 Checks. Diese Tests wurden anschließend in 10 Durchläufen untersucht, um auf Basis der erfolgreichen und fehlgeschlagenen Tests und Checks das System zu bewerten.

Von den insgesamt 390 Tests sind 28 fehlgeschlagen und 2 wurden übersprungen. Tests werden übersprungen, wenn sich die Gründe für das Fehlschlagen nicht auf das System, sondern auf die Umgebung, wie beispielsweise das Rate-Limit der GPT4o-API, beziehen.

Nach Abzug der übersprungenen Tests von der Gesamtzahl der durchgeführten Tests ergibt sich eine bereinigte Erfolgsquote von 92,78 %. Da jedoch ein Test als fehlgeschlagen gilt, sobald mindestens ein enthaltener Check fehlgeschlagen ist, ist eine ausführlichere Untersuchung der einzelnen Tests notwendig. Diese Untersuchung wurde im Folgenden auf die drei vorgestellten Test-Kategorien - Allgemein, Kartenerstellung und Testerstellung - unterteilt und die Gründe für das Fehlschlagen der Tests untersucht.

Zunächst wurden die generellen Tests untersucht. Von den insgesamt 60 Tests waren 59 erfolgreich und ein Test fehlgeschlagen. Bei Betrachtung der Checks ergibt sich, dass von den insgesamt 130 Checks ein Check fehlgeschlagen ist, welcher die Antwort des Systems prüfen sollte. Der fehlgeschlagene Test behandelt den Fall, dass der Benutzer das System nach Hilfe fragt und dieses zunächst mit dem generellen Agenten antwortet und in seiner Antwort eine Liste von Möglichkeiten enthält, wie der Benutzer Hilfe erhalten kann. In diesem fehlgeschlagenen Test antwortete das System zwar mit dem richtigen Agenten, jedoch enthielt die Antwort nicht alle Möglichkeiten, wie der Benutzer Hilfe erhalten kann.

In der zweiten Kategorie, den Ergebnissen der Karten-Tests, sind insgesamt 11 Tests von 180 Tests fehlgeschlagen und zwei weitere wurden übersprungen. Es ergibt sich hier nach Abzug der übersprungenen Tests eine bereinigte Erfolgsquote von 93,82 %. Um diese 11 Tests genauer zu untersuchen, müssen die einzelnen Checks betrachtet werden. Insgesamt besteht der Datensatz für die Karten-Tests aus 390 Checks, von denen 12 fehlgeschlagen und vier übersprungen wurden. Die bereinigte Erfolgsquote nach Abzug der übersprungenen Checks beträgt 96,69 %. Die übersprungenen zwei Tests und vier Checks sind aufgrund des bereits genannten Rate-Limits aufgetreten. In einem solchen Fall wird der Benutzer darüber in der Benutzeroberfläche informiert und kann den Prozess erneut starten. Weiter traten zwei Validierungsfehler der Antwort des LLMs bei der Überprüfung der Kartenänderung auf, insgesamt acht Fehler, in denen die Karte nicht der erwarteten Karte entsprochen hatte, und zwei Fehler, in denen die Antwort des Systems als unfreundlich eingestuft wurde. Die zwei letzteren Fehler traten auf, nachdem der Benutzer das System während des Kartenerstellungsprozesses beleidigt hatte. Die vollständige Verteilung der erfolgreichen und fehlgeschlagenen Checks auf die fehlgeschlagenen Tests ist in Abbildung 4 dargestellt.

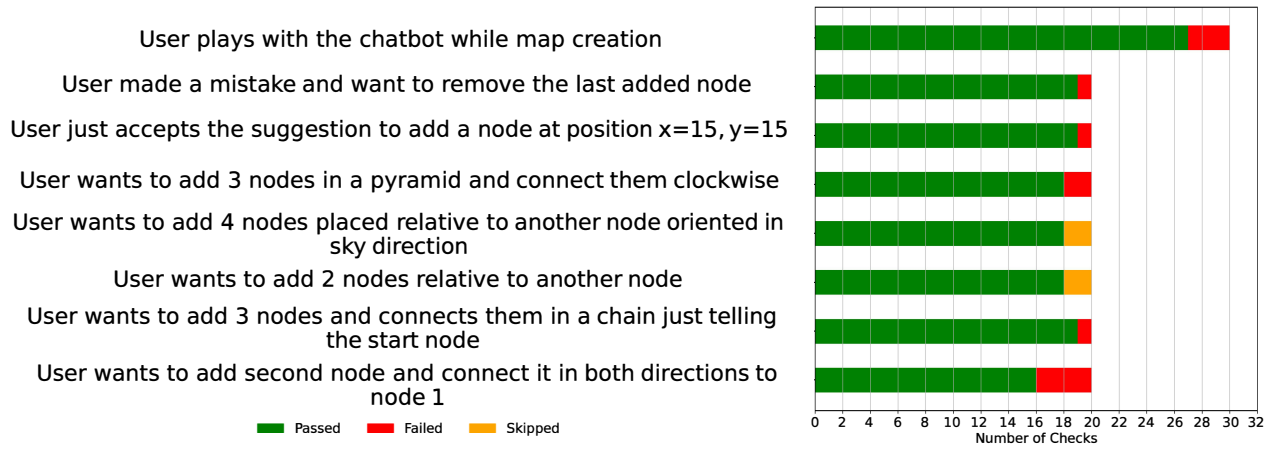

Abbildung 4: Verteilung der erfolgreichen und fehlgeschlagenen Checks in den fehlgeschlagenen Karten-Tests aus 10 Durchläufen. Zu sehen sind die acht verschiedenen Tests mit der Verteilung der erfolgreichen und fehlgeschlagenen Checks.

Quelle: Eigene Abbildung.

Die letzte Kategorie befinden sich insgesamt 150 Tests und 640 Checks. Von den 150 Tests sind 18 fehlgeschlagen, was einer Erfolgsquote von 88,00 % entspricht. Bei der Betrachtung der Checks ergibt sich, dass von den 640 Checks 24 fehlgeschlagen sind und somit eine Erfolgsquote von 96,25 %. Wie bei den vorherigen Kategorien müssen die fehlgeschlagenen 18 Tests mit 24 Checks genauer betrachtet werden. Wichtig hierbei ist, dass alleine 10 der fehlgeschlagenen Tests und 10 der fehlgeschlagenen Checks einem Testfall zuzuordnen sind. Dieser Testfall war somit in keinem der 10 Durchgänge erfolgreich. Dieser sollte das Verhalten prüfen, dass das System den zuletzt hinzugefügten Schritt entfernen sollte. Der Check, welcher den generierten Gherkin-Tests mit den zu erwartenden vergleicht, schlug jedes Mal fehl, da das System in 9 von 10 dieser Fälle zusätzlich zum zuletzt hinzugefügten Schritt auch den letzten Schritt des Gherkin-Tests entfernte. Der letzte Fall war fehlgeschlagen, da sich der Gherkin-Test nicht änderte und das System die Aufgabe nicht erfüllt hat.

Weitere fehlgeschlagene Tests waren das Entfernen einiger Schritte, in denen dem Gherkin-Test das Platzieren und Fahren der AGVs entfernt werden sollte. In drei der 10 Tests wurde der Gherkin-Test hierbei nicht korrekt angepasst, was zu drei fehlgeschlagenen Checks führt. In Folge dieser fehlerhaften Anpassung sind weitere drei Checks fehlgeschlagen, da diese prüfen sollten, ob die AGVs entfernt wurden. Dies ist allerdings nur dann möglich, wenn der Gherkin-Test keine Schritte enthält, in denen AGVs platziert oder benötigt werden.

Bei einem Test, der das Setzen eines komplexen Szenarios überprüft, trat ein weiterer Fehler auf, wodurch weitere vier Checks fehlschlugen. Das System sollte die Basis-Schritte selbstständig erzeugen, wenn der Szenario-Name ausreichend Informationen liefert. In diesem fehlgeschlagenen Test fragte das System, ob die Schritte hinzugefügt werden sollen. Da

dies definitionsgemäß keine automatische Erstellung der Basis-Schritte nach Eingabe eines Szenario-Namens mit ausreichender Information darstellt, ist dieser Test als fehlgeschlagen einzustufen. Durch das fehlende Erstellen der Basis-Schritte schlug der Check zur Prüfung des Gherkin-Tests fehl, woraufhin die darauffolgenden drei Checks, die das Vorhandensein und korrekte Platzieren der AGVs und Ladungen prüfen, ebenfalls fehlschlugen.

In den letzten vier fehlgeschlagenen Tests enthielt die Antwort des Systems nicht die ausreichenden Informationen. In einem dieser vier Tests wird ein Szenario-Name genannt, der einen AGV platziert und zu Knoten drei fahren lassen sollte. Dies ist nicht möglich, da kein Knoten drei existiert. Diesen Fehler zu erkennen und dem Benutzer entsprechende Informationen darüber zu geben, ist hierbei die Aufgabe des Systems. Aufgrund der fehlenden Informationen, dass dieses Szenario nicht möglich ist, wurde der Check und somit der Test als fehlgeschlagen eingestuft. Ähnlich wie der eben genannte Test schlugen weitere drei Tests fehl, die das Beschreiben des aktuellen Szenarios testen. Der Benutzer fragt nach der Funktionsweise des aktuellen Szenarios, woraufhin das System dieses detailliert erklären soll. In den drei fehlgeschlagenen Tests wiederholte das System lediglich den Szenario-Namen oder es fehlten Informationen, was das Szenario konkret testen soll.

Die Verteilung der erfolgreichen und fehlgeschlagenen Checks für die fehlgeschlagenen Gherkin-Tests ist in Abbildung 5 nochmals dargestellt.

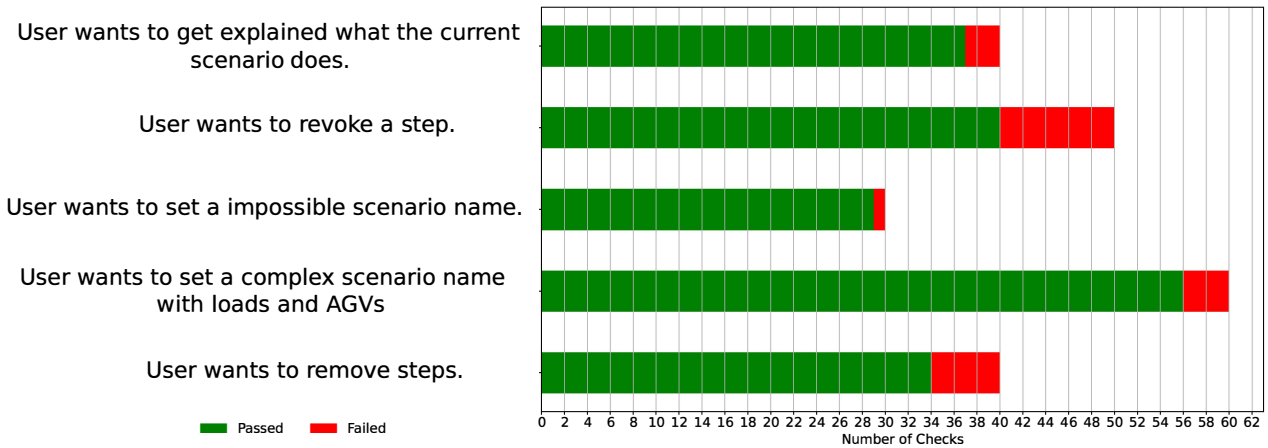

Abbildung 5: Verteilung der erfolgreichen und fehlgeschlagenen Checks in den fehlgeschlagenen Gherkin-Tests aus 10 Durchläufen. Zu sehen sind die fünf verschiedenen Tests mit der Verteilung der erfolgreichen und fehlgeschlagenen Checks.

Quelle: Eigene Abbildung.

6.2 Expertenbefragung

Der zweite Teil der Evaluation umfasst, wie erwähnt, eine Expertenbefragung mit 10 Teilnehmern. Hierdurch sollen Einblicke in die Verwendbarkeit des Systems sowie die Benutzerfreundlichkeit erfasst werden. Die Teilnehmer sollen sich aus technischen Experten der vAGV-Simulation und potentiellen Anwendern ohne tieferen technischen Hintergrund zusammensetzen. In diesem Abschnitt wird auf das Vorgehen der Expertenbefragung sowie die Ergebnisse eingegangen.

6.2.1 Vorgehen

Die Expertenbefragung beginnt zunächst mit allgemeinen Fragen zum Teilnehmer, wie den bisherigen Erfahrungen mit Chatbots oder dem Wissen über AGVs. Im zweiten Teil folgen vier Szenarien mit zu bewertenden Aussagen, wobei jedes Szenario einen bestimmten Teil der Anwendung anhand von zu lösenden Aufgaben prüft. Der letzte Teil der Expertenbefragung enthält weitere Aussagen, die zu einer abschließenden Bewertung des Systems dienen. Ein Szenario in der Befragung besteht aus einer Beschreibung mit Anweisungen oder Aufgaben und abschließenden Aussagen. Nachdem der Benutzer die Aufgaben des Szenarios erledigt hat, sollen zwischen fünf und acht Aussagen auf einer 5-Punkte-Likert-Skala[1] bewertet werden, in welcher 1 für „Ich stimme überhaupt nicht zu“ und 5 für „Ich stimme voll und ganz zu“ steht. Die 5-Punkte-Likert-Skala wurde aufgrund der Empfehlung von Abeysinghe et al. gewählt. Abeysinghe et al. verwendeten die 5-Punkte-Likert-Skala zur Bewertung der Antworten von Chatbot, diese wird in der vorliegenden Arbeit jedoch zur Bewertung der Benutzererfahrung verwendet [41, S. 1ff.].

Die Likert-Skala ermöglicht dabei verschiedene Aspekte, wie das Verständnis und die Verwendbarkeit zu bewerten, wodurch Einblicke in die Benutzerfreundlichkeit des Systems gewonnen werden können. Die verwendeten Aussagen sind im Anhang B zu finden. Im Folgenden werden die vier Szenarien beschrieben.

1. **Szenario: Erster Eindruck**

   Das erste Szenario soll Aufschluss über die ersten Eindrücke auf der Benutzeroberfläche liefern. Dies beinhaltet unter anderem die Begrüßungsnachricht des Chatbots sowie

[1] Die Likert-Skala ist eine psychometrische Skala, die aus mehreren Aussagen besteht, welche auf Ratingskalen hinsichtlich des Grades der Zustimmung einzuschätzen sind [43, S.292].

die Verständlichkeit, was der erste Schritt bei der Verwendung des Chatbots ist. Hierzu werden dem Teilnehmer abschließend fünf zu beurteilenden Aussagen vorgesetzt.

2. **Szenario: Kennenlernen**

   Im zweiten Szenario soll der Teilnehmer mit den ersten grundlegenden Funktionen vertraut gemacht werden. Hierzu werden fünf Aufgaben gestellt, die das Erstellen einer Karte mit dem Hinzufügen, Entfernen und Ändern von vorgegebenen Knoten und Kanten erfordern. Der Teilnehmer soll hierbei die einzelnen Aufgaben lösen und mit dem Chatbot interagieren. In den letzten zwei Aufgaben soll der Chatbot befragt werden, wie die Karte exportiert werden kann und dann die Karte für ein späteres Szenario heruntergeladen werden.

3. **Szenario: Karte ohne Anleitung erstellen**

   Nachdem der Teilnehmer erste Kartenelemente erstellt und geändert hat, wird im dritten Szenario lediglich ein Bild einer bestehenden Karte dargestellt mit der Anweisung, diese Karte mit dem Chatbot zu erstellen. Der Teilnehmer wird nach dem Erstellen der Karte zunächst nach der Anzahl an Nachrichten gefragt und soll dann mit der Beurteilung von fünf Aussagen fortfahren.

4. **Szenario: Gherkin Test erstellen**

   Da sich die Funktionen des Chatbots nicht nur auf die Kartenerstellung beschränken wird der Teilnehmer im vierten Szenario mit der Testerstellung bekannt gemacht. Dieses Szenario teilt sich in zwei Abschnitte und behandelt zunächst die grundlegenden Funktionen wie das Starten der Testerstellung, Benennen des Features und Test-Szenarios sowie das Erklären lassen des erstellten Tests. Im zweiten Teil des Szenarios werden dem Teilnehmer weitere Aufgaben zum Ändern des Gherkin-Tests gestellt. Zum Abschluss von jedem Teil wird eine Liste aus Aussagen zur Beurteilung vorgelegt.

Zusätzlich haben die Teilnehmer die Möglichkeit, Kommentare in einem dafür vorgesehenen Freitextfeld abzugeben. Die Kommentare der Teilnehmer werden im Anschluss inhaltlich ausgewertet, um weitere Einblicke zu sammeln und Verbesserungsvorschläge zu identifizieren. Die Durchführung der Befragung wird persönlich mit den Teilnehmenden erfolgen, um im Nachgang einen Dialog und eine Diskussion zu ermöglichen.

6.2.2 Ergebnisse

Die Expertenbefragung wurde mit insgesamt 10 Teilnehmern, bestehend aus technischen Experten und potenziellen Anwendern, durchgeführt. Durchschnittlich wurden 4,65 von 5 Punkten für das System aus allen Szenarien gegeben. Im Folgenden werden die Ergebnisse je Szenario untersucht sowie die von den Teilnehmern angemerkten Kommentare beschrieben.

Für das erste Szenario wurden durchschnittlich 4,72 von 5 Punkten für alle Aussagen vergeben. Die Antworten bewegten sich, bis auf eine Ausnahme, in der drei Punkte vergeben wurden, im Bereich zwischen vier und fünf Punkten. Neben der Bewertung der Aussagen wurden von drei Teilnehmern Kommentare und Verbesserungsvorschläge beschrieben. Es wurde angemerkt, dass der Zweck der zu Beginn leeren Kartenkomponente nicht direkt erkennbar ist, der Tour-Button eher einem Help-Button entspricht, das in der Erklärung erwähnte Kartenformat LIF möglicherweise nicht jedem bekannt ist und die Buttons unterhalb des Begrüßungstextes nicht unbedingt als anklickbare Buttons erkannt werden könnten. Zudem wurde vorgeschlagen, das System in mehreren Sprachen, insbesondere Deutsch, verfügbar zu machen. Die Ergebnisse der einzelnen Aussagen sind in Abbildung 6 dargestellt.

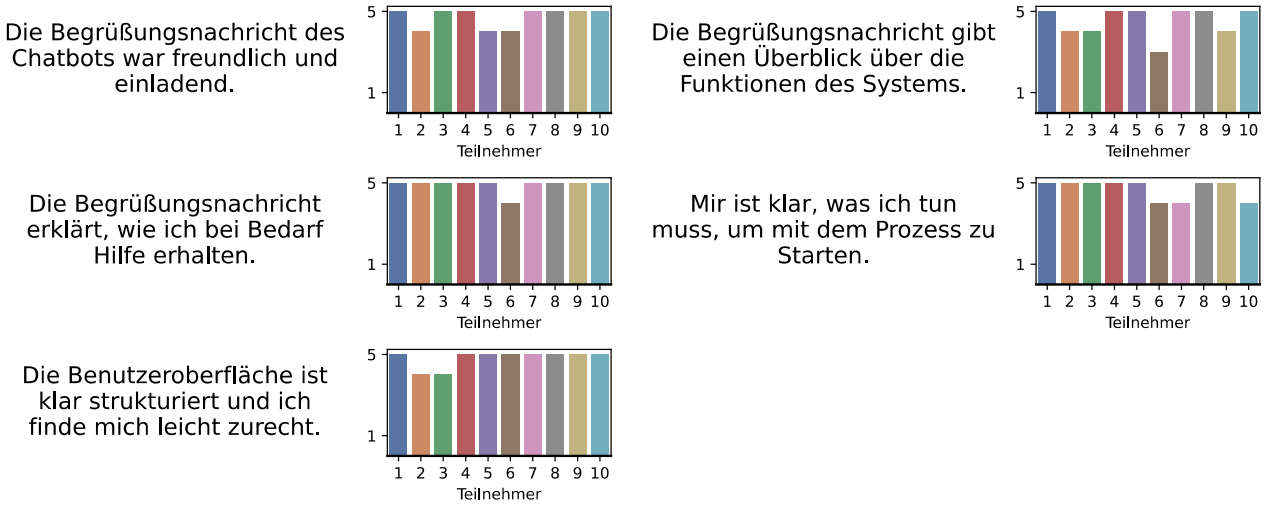

Abbildung 6: Auswertung der Expertenbefragung von Szenario 1 „Erster Einblick“ . Zu sehen sind die gestellten Aussagen mit dazugehörigen Antworten der Teilnehmer.

Quelle: Eigene Abbildung.

Wie für das erste Szenario wurden im Durchschnitt für das Zweite 4,72 von 5 Punkten für alle Aussagen vergeben. Auffällig ist hierbei, dass sich bei der Aussage, welche sich dem Herunterladen der Karte widmet, von einem Teilnehmer ein Punkt vergeben wurde. Dieser Teilnehmer beschrieb in der nachfolgenden Diskussion die Schwierigkeit, den Download-Button als Button zu identifizieren. Diese Schwierigkeit wurde im Verlauf weiterer Befragungen erneut

angemerkt. Des Weiteren wurde bei einem Teilnehmer eine nicht gewünschte Verbindung hinzugefügt, welche der Teilnehmer durch eine Korrekturantwort entfernen konnte.

Neben diesen Problemen und der Bewertung der Aussagen wurden von weiteren Teilnehmern Kommentare und Verbesserungsvorschläge beschrieben. Hinsichtlich der integrierten Karte wurde kritisiert, dass die Skalierung der Karte nicht optimal ist und die verwendeten Positionen nur vom Chatbot paraphrasiert, aber nicht in der Karte visualisiert werden. Zusätzlich beschrieben die Teilnehmer die vom Chatbot bereitgestellten Vorschläge teilweise als nicht hilfreich oder nicht sinnvoll. Weiter wurde der Wunsch geäußert, die vorherigen geschriebenen Benutzerantworten über die Pfeiltasten auf der Tastatur in Anlehnung an die Funktion eines Terminals auswählen zu können. In Abbildung 7 sind alle Aussagen aus dem zweiten Szenario mit der dazugehörigen Verteilung der Teilnehmerantworten zu sehen.

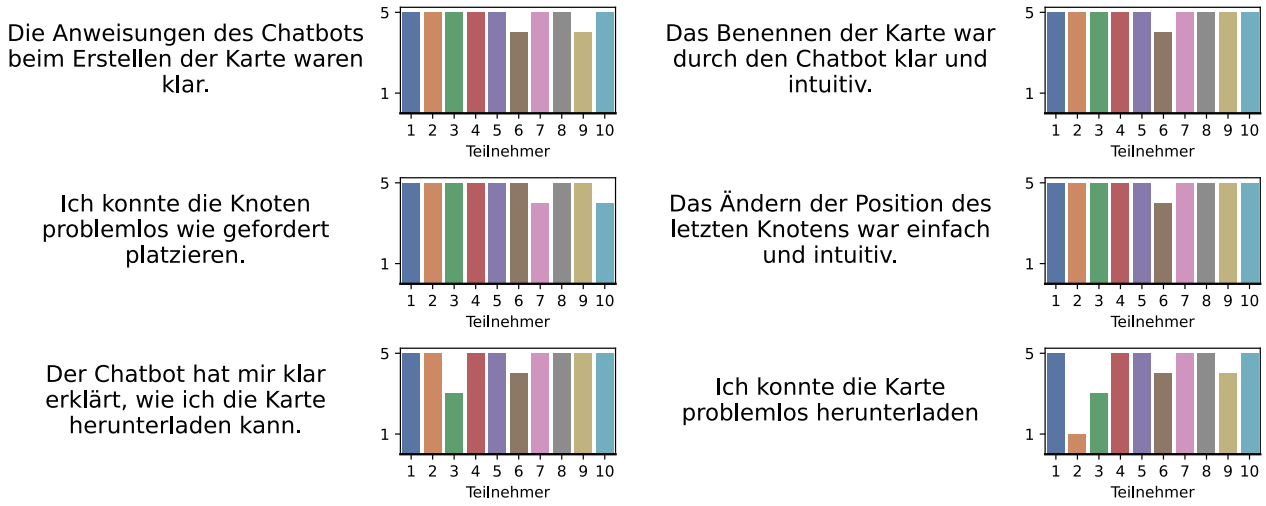

Abbildung 7: Auswertung der Expertenbefragung von Szenario 2 „Kennenlernen“ . Zu sehen sind die gestellten Aussagen mit dazugehörigen Antworten der Teilnehmer.

Quelle: Eigene Abbildung.

Die Aussagen im dritten Szenario wurden durchschnittlich mit 4,34 von 5 Punkten bewertet, wobei sowohl Ein-Punkte-Bewertungen als auch Fünf-Punkte-Bewertungen für dieselbe Aussage vergeben wurden. Bis auf einen Teilnehmer, bei dem der Chatbot die Anweisungen nicht korrekt umgesetzt hatte und nicht auf Korrekturvorschläge eingegangen war, konnten alle die geforderte Karte erstellen. Die Anzahl der benötigten Nachrichten variierte hierbei von zwei bis 21 Nachrichten, wobei der Median bei 4,5 Nachrichten lag. Es wurde angemerkt, dass das Unterteilen der Anweisungen in mehrere kleine Anfragen besser funktionierte, als das komplette Ziel in einer Nachricht zu beschreiben. Zusätzlich wurden Verbesserungsvorschläge beschrieben, wie das Benennen der exportierten Kartendatei basierend auf dem Namen der Karte, das manuelle Verschieben einzelner Knoten innerhalb der Karte sowie das

Anlegen einer Übersichtsliste oder das Beschriften der Koordinaten der Knoten innerhalb der Karte mit einem Raster. Die Verteilung der einzelnen Aussagen ist in Abbildung 8 dargestellt.

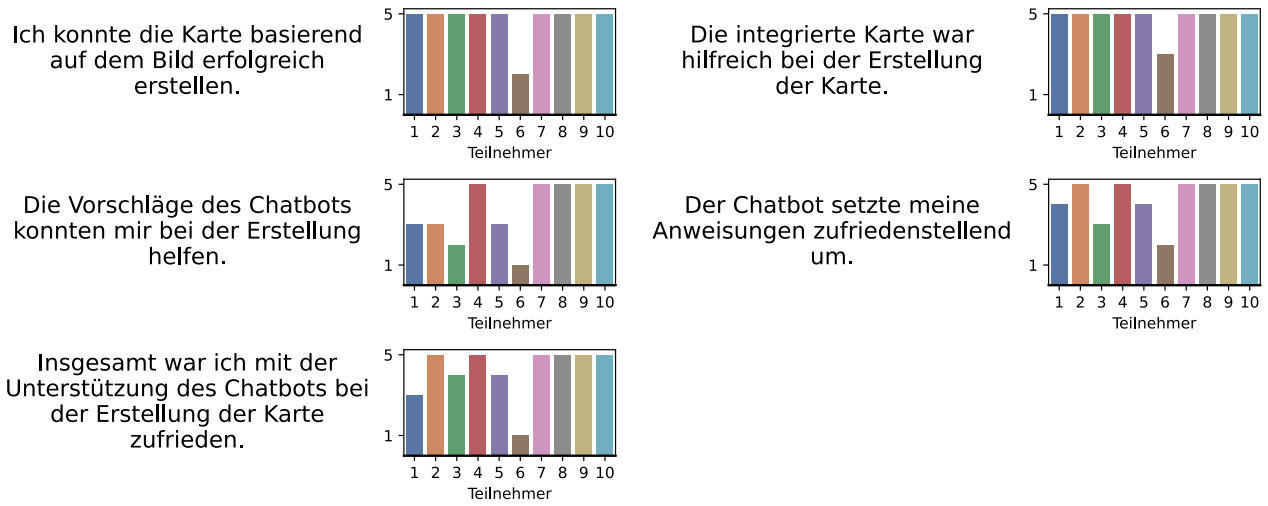

Abbildung 8: Auswertung der Expertenbefragung von Szenario 3 „Karte ohne Anleitung erstellen“ . Zu sehen sind die gestellten Aussagen mit den dazugehörigen Antworten der Teilnehmer.

Quelle: Eigene Abbildung.

Bei dem vierten Szenario wurde der erste Teil, welcher sich der Erstellung eines eigenen Gherkin-Tests widmet, mit durchschnittlich 4,70 von 5 Punkten bewertet. Wie im dritten Szenario bewegen sich hierbei die Bewertungen zwischen eins und fünf für dieselbe Aussage. Diese Ein-Punkte-Bewertung ist auf einen internen Fehler der Simulation zurückzuführen, wodurch kein AGV in der Karte angezeigt werden konnte und somit die Aussage, ob dieses beim Verständnis unterstützen würde, mit einem Punkt bewertet wurde. Bei den restlichen Aussagen bewegen sich die Bewertungen zwischen drei und fünf Punkten, wobei jeder Teilnehmer in der Lage war, den beschriebenen Gherkin-Test zu erstellen. Dennoch wurde von einem Teilnehmer in der nachfolgenden Diskussion beschrieben, dass der Chatbot zunächst nicht nach einem Szenario-Namen fragte und dies zu Verwirrungen führte.

Die Kommentare und Verbesserungsvorschläge umfassen den Wunsch nach einem „Play-Button“ für die direkte Ausführung des Gherkin-Tests mit einem parallelen Highlighten des aktuell ausgeführten Gherkin-Schrittes. Zusätzlich wurde die Funktion, die Karte mithilfe von Drag-and-Drop einzufügen, sowie die Möglichkeit, die Karte bei der Testerstellung kleiner zu ziehen, um den Fokus auf den Gherkin-Test zu legen, geäußert. Ein festgestelltes Problem, dass die Position des AGVs in der Karte unklar ist, wurde beschrieben und der Wunsch nach einer Erklärung nach dem Platzieren des AGVs geäußert. Als letzter Kommentar wurde angemerkt, dass der Chatbot nach dem Hochladen einer Karte erneut nach einem

Kartennamen fragt, was kurzzeitig für Verunsicherung sorgte. Die Aussagen mit Bewertungen sind in Abbildung 9 dargestellt.

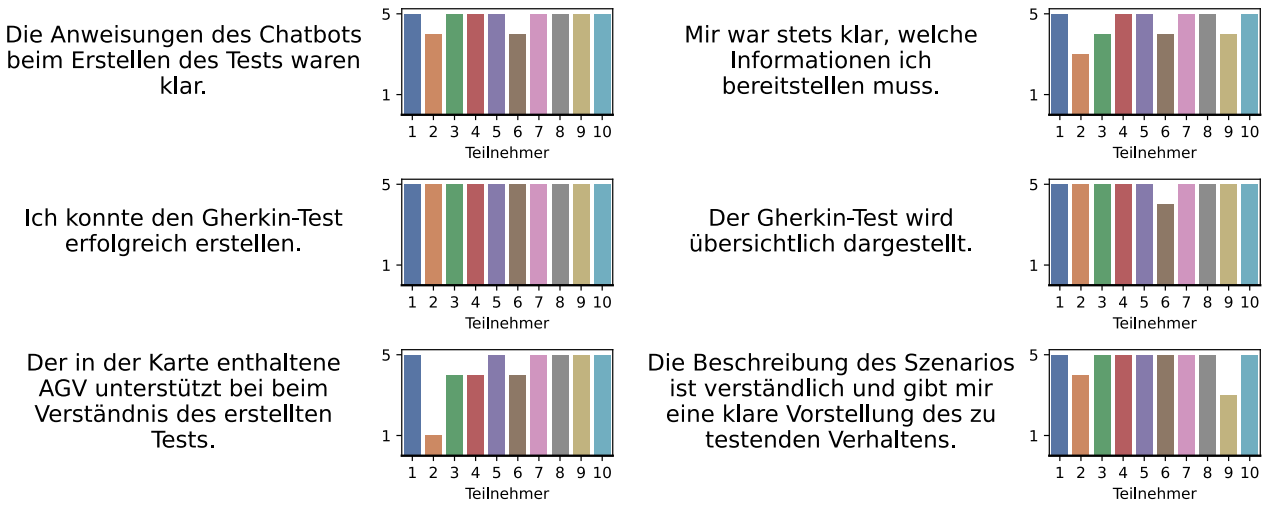

Abbildung 9: Auswertung der Expertenbefragung von Szenario 4 Teil 1 „Gherkin Test erstellen“ . Zu sehen sind die gestellten Aussagen mit dazugehörigen Antworten der Teilnehmer.

Quelle: Eigene Abbildung.

Im zweiten Teil des vierten Szenarios, in welchem der erstellte Test aus dem ersten Teil angepasst werden sollte, wurde durchschnittlich mit 4,73 von 5 Punkten bewertet. Es wurde angemerkt, dass der Chatbot beim Hinzufügen eines Schritts erneut forderte, dass der Schritt hinzugefügt werden soll, obwohl er bereits vorhanden war, der Chatbot über mehrere Nachrichten hinweg einen Fehler aus einer vorherigen Frage wiederholt hatte oder der entfernte Schritt nach dem Umbenennen des Szenarios erneut im Test enthalten war.

Es wurde vorgeschlagen, statt der Keywords, welche ein Fenster mit der Erklärung des einzelnen Gherkin-Schrittes öffnen, die gesamte Testzeile klickbar zu machen, um die Erklärung zu öffnen. Weiter sollte der Name der Karte in der Benutzeroberfläche angezeigt werden und der Benutzer über die Folgen beim Löschen eines Schrittes oder Umbenennen des Szenarios benachrichtigt werden. Der letzte Punkt bezieht sich hierbei auf die Funktion, dass die internen Abhängigkeiten einzelner Gherkin-Schritte beim Löschen möglicherweise ebenfalls entfernt werden oder beim Umbenennen des Szenarios die Schritte passend zum neuen Szenario-Namen angepasst werden. Das bereits im ersten Szenario erwähnte Navigieren mit den Pfeiltasten zur Auswahl der vorherigen Nachrichten wurde erneut von einem weiteren Teilnehmer als Wunsch geäußert, gemeinsam mit der Möglichkeit, einzelne Zeilen des Tests auszukommentieren oder zu entfernen, ohne eine Nachricht an den Chatbot senden zu müssen. Die Verteilung der Antworten für die Aussagen ist in Abbildung 10 zu sehen.

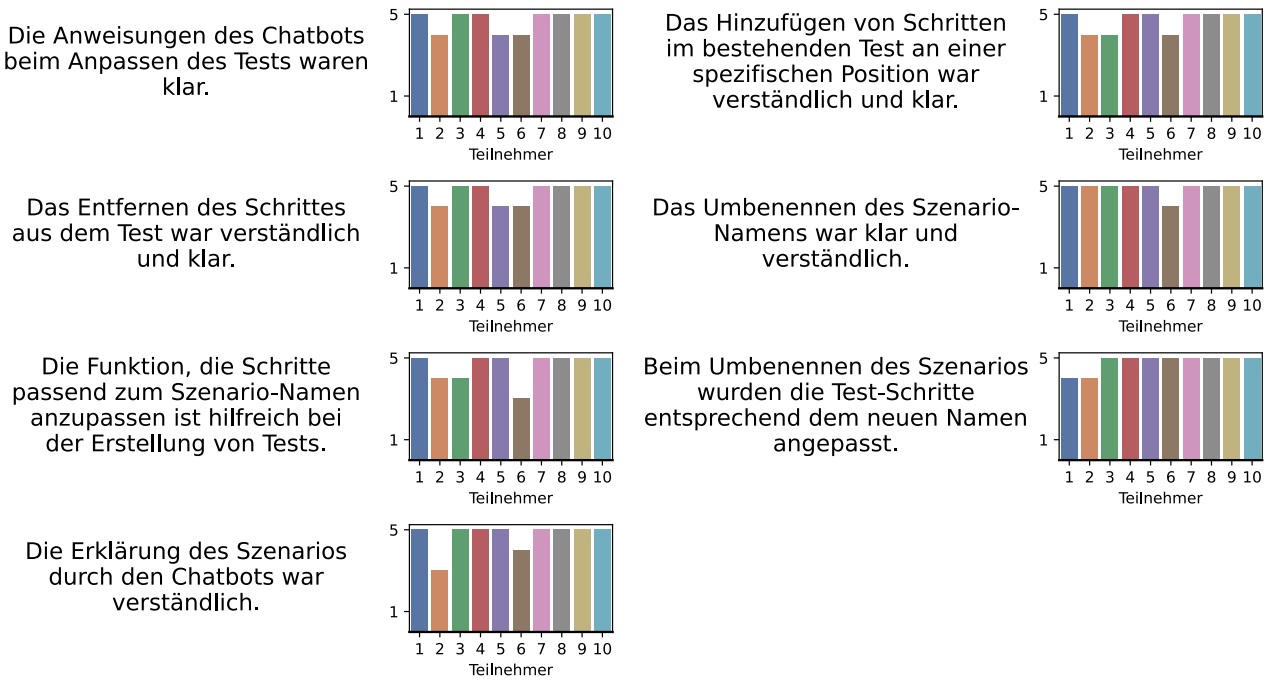

Abbildung 10: Auswertung der Expertenbefragung von Szenario 4 Teil 2 „Gherkin Test anpassen “ . Zu sehen sind die gestellten Aussagen mit dazugehörigen Antworten der Teilnehmer.

Quelle: Eigene Abbildung.

Für die abschließenden Fragen wurden durchschnittlich 4,73 von 5 Punkten vergeben und von einem Teilnehmer Kommentare angemerkt, die sich der Verwendbarkeit des Systems in einer produktiven Umgebung widmen. Der Teilnehmer merkte an, dass sich das System erst bei größeren, komplexeren Gherkin-Tests auszahlt, da bei kleineren Tests die Wartezeiten und das Korrigieren von Fehlern länger dauern könnten, als das händische Schreiben des Tests. Zudem wurden die Grenzen eines LLMs erwähnt, welche zugleich die Grenzen des Systems repräsentieren. Die Aussagen und Bewertungen sind in Abbildung 11 dargestellt.

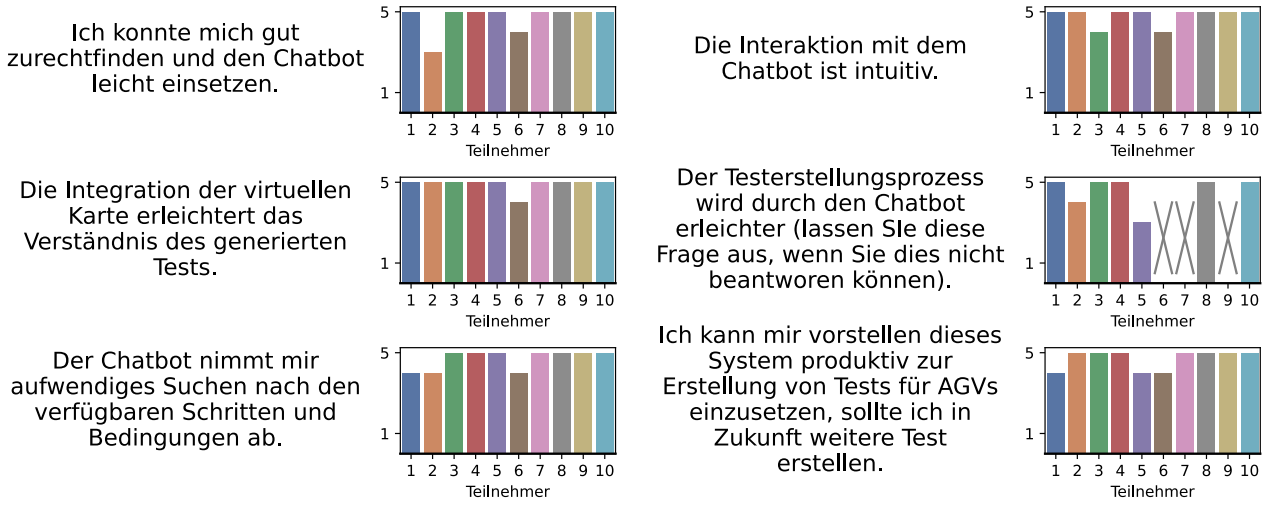

Abbildung 11: Auswertung der abschließenden Aussagen der Expertenbefragung. Zu sehen sind die gestellten Aussagen mit den dazugehörigen Antworten der Teilnehmer.

Quelle: Eigene Abbildung.

7 Diskussion

Dieses Kapitel befasst sich mit der Diskussion der Ergebnisse aus der Validierung. In den folgenden Abschnitten werden die Ergebnisse aus Kapitel 6 diskutiert und die Kommentare und Verbesserungsvorschläge von den Teilnehmern interpretiert.

Die im Kapitel 1 erwähnten zentralen Forschungsfragen dieser Arbeit lauten:

Q1 Wie verlässlich können mithilfe eines LLM-basierten Chatbots valide Gherkin-Tests und LIF-Karteninformationen für AGVs anhand von Chat-Anweisungen erstellt und modifiziert werden?

Q2 Wie benutzerfreundlich ist die Erstellung und Anpassung von Gherkin-Tests und LIF-Karteninformationen für AGVs mithilfe eines LLM-basierten Chatbots?

Zur Beantwortung der ersten Forschungsfrage Q1 dient die im Abschnitt 6.1 vorgestellte testbasierte Evaluation. Die Ergebnisse dieser Evaluation zeigten vereinzelt Fehler, die teilweise auf das System und teilweise auf die Umgebung zurückzuführen sind. Die Fehler werden im Folgenden interpretiert.

Der in den allgemeinen Tests aufgetretene fehlerhafte Test, in welchem die Liste möglicher Anlaufstellen wie der Benutzer Hilfe erhalten kann, nicht in der Antwort des Chatbots enthalten war, wurde im Nachgang nochmals detailliert untersucht. Der Chatbot antwortete mit einer Nachricht, die den Benutzer auffordert, weitere Informationen über die Probleme zu liefern, um gemeinsam eine Lösung zu finden. In einer produktiven Verwendung des Systems könnte dies dem Benutzer denselben Mehrwert wie die Auflistung aller verfügbaren Anlaufstellen liefern. Die allgemeinen Tests können somit als voller Erfolg eingestuft werden.

Neben den allgemeinen Tests wurden die fehlgeschlagenen Tests der Kartenerstellung und Kartenmodifikation nachträglich untersucht. Die aufgetretenen Validierungsfehler beziehen sich nicht auf ein systeminternes Problem, sondern vielmehr auf ein Problem, dass das verwendete LLM trotz der Verwendung klar definierter JSON-Schemas die Antworten teils unvollständig liefert und daraufhin mit der verwendeten Bibliothek Pydantic nicht geparst werden kann. Trotz einiger Abfangmechanismen, wie der bis zu dreimaligen Wiederholung der Anfrage beim Auftreten des Fehlers, kann keine einhundertprozentige Sicherheit hergestellt werden. Bei Auftreten des Fehlers wird dem Benutzer eine entsprechende Fehlermeldung in der Benutzeroberfläche angezeigt, woraufhin der Benutzer den Prozess erneut starten kann. Ein Update des verwendeten LLMs soll dieses Problem durch die Einführung der

sogenannten Structured-Outputs[1] beheben. Da zum Zeitpunkt der Arbeit dieses LLM nicht verwendet werden konnte, wurde dies jedoch nicht getestet.

In den Fällen, in denen fehlerhafte Karteninformationen erstellt wurden, wurden die Anweisungen nicht korrekt umgesetzt. Eine detaillierte Untersuchung der Chatbot-Antworten zeigte, dass der Chatbot teilweise nur eine der geforderten Aktionen durchführte und für die zweite Aktion eine Rückfrage stellte. In anderen Fällen behauptete der Chatbot, die Aktion ausgeführt zu haben, jedoch wurde die Karte nicht angepasst. Weitere Anpassungen am Prompt könnten die Häufigkeit des Auftretens dieses Problems verringern, jedoch ist bei der Verwendung von LLMs ein gewisses Maß an Ungenauigkeit nicht vollständig ausschließbar.

Die letzte Fehlerkategorie der Kartentests bezieht sich auf die Freundlichkeit der Nachrichten, wobei zwei der fehlgeschlagenen Checks durch ein LLM-as-Judge-Verfahren fälschlicherweise als fehlgeschlagen eingestuft wurden. Die Nachrichten des Chatbots wurden hier als fehlgeschlagen eingestuft, da diese als zu formell oder nicht freundlich genug eingestuft wurden, jedoch nach erneuter Untersuchung durch einen Menschen als freundlich eingestuft werden können.

Bei den fehlgeschlagenen Gherkin-Tests zeigt sich deutlich, dass ein im Rahmen der Validierung beschriebener Test durch das System bislang nicht erfolgreich durchgeführt werden konnte. Die fehlerhafte Interpretation der Benutzeranfrage seitens des Chatbots lässt sich möglicherweise durch weitere Anpassungen am Prompt und durch das Integrieren eines Beispiels verbessern. Neben diesen Tests lassen sich weitere fehlgeschlagene Tests, die das Entfernen einzelner Schritte behandelten, auf dasselbe Problem zurückführen und könnten durch eine Anpassung des Prompts behoben werden.

Weiter trat beim Benennen des Szenarios ein Fehler auf, da die definitionsgemäß geforderten Basis-Schritte nicht automatisiert gesetzt wurden. Eine detaillierte Untersuchung dieses Tests zeigte jedoch, dass der Chatbot vor dem automatisierten Hinzufügen der Schritte eine Bestätigung durch den Benutzer anforderte. Dies kann somit teilweise als Erfolg eingestuft werden. Eine Aussage, ob die Schritte nach der Bestätigung durch den Benutzer erfolgreich und vollständig hinzugefügt wurden, kann jedoch nicht getroffen werden.

Die Tests zur Prüfung einer ausführlichen Erklärung des Szenarios sowie das Erkennen von nicht zulässigen Szenarios zeigten Schwierigkeiten, die auf den Prompt zurückzuführen

[1] Weitere Informationen zu Structured-Outputs können unter https://platform.openai.com/docs/guides/structured-output nachgelesen werden.

sind. Dies führt ebenfalls zu der Schlussfolgerung, weitere Anpassungen am Prompt durch konkrete Beispiele oder ausführlichere Beschreibungen der einzelnen Fälle vorzunehmen.

Abschließende Beantwortung der Forschungsfrage Q1: Die Erstellung und Modifikation von validen Gherkin-Tests und LIF-Karteninformationen für AGVs mithilfe eines LLM-basierten Chatbots kann als verlässlich eingestuft werden. Die testbasierte Evaluation ergab eine bereinigte Erfolgsquote von 92,78 %, was auf eine hohe Verlässlichkeit hinweist. Obwohl einige Fehler und Probleme identifiziert wurden, könnten diese voraussichtlich durch weitere Anpassungen des Prompts und Optimierungen des Systems verringert oder behoben werden. Die positiven Ergebnisse der allgemeinen Tests und die Fähigkeit der Teilnehmer, die meisten Aufgaben erfolgreich zu bearbeiten, bestätigen die Verlässlichkeit des Systems.

Um die zweite Forschungsfrage Q2 beantworten zu können, dienen die Ergebnisse der Expertenbefragung aus Abschnitt 6.2, welche tiefere Einblicke in die Verwendbarkeit und Benutzerfreundlichkeit des Systems liefern. Die Szenarien untersuchten die Grundfunktionen des Systems, und durch die Diskussionen mit den Teilnehmern im Anschluss an die Befragung konnten Verbesserungsvorschläge und Kommentare der Teilnehmer besprochen werden. Im Folgenden werden die Probleme und Verbesserungsvorschläge genauer betrachtet.

Die von den Teilnehmern angesprochenen Probleme sind teilweise auf dieselben Ursachen wie die in den vorherigen Abschnitten beschriebenen Fehler zurückzuführen. Beispielsweise reagierte der Chatbot nicht auf Korrekturvorschläge, fragte nicht nach den benötigten Funktionen oder lieferte unklare und nicht hilfreiche Vorschläge. Diese Probleme traten jedoch in der Expertenbefragung nur vereinzelt auf und hinderten die Teilnehmer bis auf wenige Ausnahmen nicht an der Bearbeitung der Aufgaben. Zur Verbesserung könnten weitere Anpassungen des Prompts dienen, jedoch können diese Probleme aufgrund der Unsicherheiten bei der Arbeit mit LLMs nicht vollständig ausgeschlossen werden.

Einige Probleme und Wünsche bezogen sich auf die Benutzeroberfläche. Hierbei muss jedoch unterschieden werden, ob sich die Probleme und Wünsche auf Verbesserungen des im Rahmen der vorliegenden Arbeit entwickelten Systems oder die vAGV-Simulation beziehen. Mehrere Teilnehmer merkten Probleme und Wünsche an, die die integrierte Karte betreffen. Diese wird zum aktuellen Zeitpunkt lediglich von der Simulation integriert, wodurch kein Einfluss auf die Skalierung, das Anzeigen eines Rasters oder die Integration von Koordinaten neben den Knoten genommen werden kann. Die genannten Probleme und Verbesserungen beziehen sich auf die verwendete vAGV-Simulation und können im Rahmen

einer Anpassung des Chatbot-Systems nicht direkt gelöst werden. Eine Alternative wäre, eine eigene Variante der Karte aus den LIF-Karteninformationen zu erstellen, in welcher Positionen und Raster angezeigt werden, Knoten manuell verschoben und benutzerdefinierte Skalierungen angewendet werden können.

Andere Probleme und Wünsche bezogen sich jedoch nicht ausschließlich auf die Karte. Probleme wie das Identifizieren der Buttons können durch Anpassungen der Benutzeroberfläche behoben werden, um diese klar von Überschriften und anderen Komponenten abzuheben. Weiter wurde angemerkt, dass die Position des AGVs in der Karte erklärt werden sollte. Hierzu könnte beim Platzieren des AGVs ein Pop-up-Fenster geöffnet werden, das die aktuelle Position erklärt. Auch der vergebene Name für die Karte könnte innerhalb der Kartenkomponente angezeigt und für das Benennen der exportierten Kartendateien verwendet werden. Zusätzlich zu den kartenbezogenen Wünschen wurden weitere geäußert, die zu einer verbesserten Benutzerfreundlichkeit führen könnten. Die Auswahl vorheriger Nachrichten über die Pfeiltasten der Tastatur in Anlehnung an ein Terminal wurde mehrfach gewünscht, da teilweise lediglich Änderungen in der vorherigen Nachricht vorgenommen werden sollen und diese somit nicht händisch kopiert und eingefügt werden müssen. Dies umzusetzen benötigt allerdings größere Anpassungen im Frontend sowie Backend und die Integration eines weiteren Endpunktes, der die Liste der vorherigen Nachrichten liefert.

In Bezug auf den Gherkin-Test und die in der Benutzeroberfläche angezeigten Gherkin-Schritte wurde der Wunsch geäußert, den Fokus beim Erstellen des Tests auf die Komponente, in welcher die Schritte dargestellt werden, zu verschieben und die angezeigte Kartenkomponente zu verkleinern. Dies könnte durch Frontend-Anpassungen realisiert werden, jedoch besteht hierbei weiterhin das zuvor genannte Problem, aktuell keinen Einfluss auf die dargestellte Karte zu haben. Weiter wurde der Wunsch geäußert, einen Play-Button zur direkten Ausführung des Gherkin-Tests in der Benutzeroberfläche zu integrieren. Diese Idee wurde bereits bei der Implementierung untersucht, gestaltet sich jedoch unter anderem durch die fehlende Simulations-Schnittstelle zur Ausführung der Gherkin-Tests über eine API schwierig. Eine solche Funktionalität benötigt die enge Zusammenarbeit mit den Entwicklern der vAGV-Simulation.

Neben diesen Wünschen bezogen sich andere direkt auf die dargestellten Gherkin-Schritte. Der Wunsch, nicht länger ausschließlich durch das Klicken der Gherkin-Keywords die dazugehörigen Schritt-Erklärungen zu öffnen, könnte durch Anpassungen im Frontend realisiert werden, indem die gesamte Zeile klickbar gemacht wird. Weiter wurde der Wunsch geäußert,

einzelne Zeilen auszukommentieren oder zu entfernen, ohne eine Nachricht an den Chatbot senden zu müssen. Dies benötigt zusätzlich einen weiteren Endpunkt im Backend, wodurch einzelne Zeilen aus dem generierten Gherkin-Test entfernt werden können. Allgemein wurde zusätzlich der Wunsch geäußert, das System in mehreren Sprachen, insbesondere Deutsch, zur Verfügung zu stellen, den Tour-Button anzupassen und das LIF-Kartenformat durch die Integration eines Links zur offiziellen Dokumentation erklärbar zu machen.

Neben Problemen und Wünschen zeigte sich jedoch, dass jeder Teilnehmer die Aufgaben größtenteils bearbeiten konnte und das System durch eine durchschnittliche Bewertung von 4,65 von 5 Punkten aus allen Szenarien und Aussagen positiv aufgenommen wurde. Die abschließenden Fragen, welche sich direkt auf die zweite Forschungsfrage Q2 beziehen, zeigten zudem, dass sich jeder Teilnehmer gut zurechtfinden konnte und sich vorstellen kann, das System in Zukunft zur Erstellung von Gherkin-Tests für AGVs zu verwenden.

Abschließende Beantwortung der Forschungsfrage Q2: Die Erstellung und Anpassung von Gherkin-Tests und LIF-Karteninformationen für AGVs mithilfe eines LLM-basierten Chatbots wurde von den Teilnehmern der Expertenbefragung als benutzerfreundlich bewertet. Trotz einiger identifizierter Probleme und Verbesserungsvorschläge konnte das System insgesamt überzeugen und erhielt eine hohe durchschnittliche Bewertung. Die Teilnehmer konnten die Aufgaben größtenteils erfolgreich bearbeiten und waren zufrieden mit der Benutzerfreundlichkeit des Systems. Die gesammelten Kommentare und die positive Bewertung bestätigen, dass das System eine sinnvolle Unterstützung bei der Erstellung und Anpassung von Gherkin-Tests und LIF-Karteninformationen bietet.

8 Zusammenfassung und Ausblick

In dieser Arbeit wurde ein LLM-basierter Chatbot entwickelt und hinsichtlich der Verlässlichkeit und Benutzerfreundlichkeit bei der Erstellung und Anpassung von Gherkin-Tests und LIF-Karteninformationen für AGVs untersucht. Hierbei lieferte die entwickelte testbasierte Evaluation wertvolle Einblicke in die Stärken und Schwächen des Systems, und die durchgeführte Expertenbefragung gab tiefere Einblicke in die Benutzbarkeit und Benutzerfreundlichkeit.

Die erste Forschungsfrage Q1, die sich mit der Verlässlichkeit des Systems befasste, konnte durch die testbasierte Evaluation beantwortet werden. Mit einer bereinigten Erfolgsquote von 92,78 % wurde gezeigt, dass der Chatbot in der Lage ist, valide Gherkin-Tests und LIF-Karteninformationen zuverlässig zu erstellen und anzupassen. Trotz vereinzelter Fehler und Probleme, die hauptsächlich auf eine fehlerhafte Interpretation der Benutzeranfragen und damit verbundene fehlerhafte Verarbeitung der Anfrage durch das LLM zurückzuführen sind, konnte das System insgesamt überzeugen. Weitere Anpassungen des Prompts und Optimierungen des Systems könnten die Verlässlichkeit weiter erhöhen.

Die zweite Forschungsfrage Q2, die die Benutzerfreundlichkeit des Systems untersuchte, wurde durch die Expertenbefragungen beantwortet. Die 10 Teilnehmer bewerteten die in der Befragung enthaltenen Aussagen über das System mit durchschnittlich 4,65 von 5 Punkten, was auf eine hohe Zufriedenheit hinweist. Fast alle Teilnehmer konnten die Aufgaben erfolgreich bearbeiten und äußerten positive Rückmeldungen zur Bedienbarkeit des Systems. Die von den Teilnehmern geäußerten Kommentare und Verbesserungsvorschläge bezogen sich hauptsächlich auf die Benutzeroberfläche und die Interaktion mit dem Chatbot, die durch zukünftige Anpassungen weiter optimiert werden können.

Die Erkenntnisse aus dieser Arbeit ergeben mehrere Ansätze für zukünftige Verbesserungen:

- **Optimierung des Prompts:** Durch weitere Anpassungen und Verbesserungen des Prompts könnten die Genauigkeit und Verlässlichkeit des Chatbots weiter erhöht werden. Insbesondere könnte das Integrieren von Beispielen zu den identifizierten Schwierigkeiten im Prompt die richtige Interpretation der Benutzeranfragen verbessern.
- **Verbesserung der Benutzeroberfläche:** Anpassungen und Erweiterungen der Benutzeroberfläche, wie das Hervorheben der anklickbaren Buttons oder das Anzeigen von Erklärungen durchgeführter Aktionen in Pop-up-Fenstern, könnten die Bedienbarkeit des Systems weiter verbessern.

- **Erweiterung der verfügbaren Systemsprachen:** Die Benutzerfreundlichkeit für nicht-englischsprachige Anwender könnte durch die Erweiterung der Systemsprachen, primär Deutsch, verbessert werden sowie die Akzeptanz des Systems steigern.
- **Nutzung des Structured-Outputs:** Ein Update des verwendeten LLMs, welches den erwähnten Structured-Output-Mode verwendet, könnte zu weniger Validierungsfehlern durch die verwendete Bibliothek Pydantic führen und so die Geschwindigkeit des Systems durch weniger Fehlerbehandlungen verbessern.
- **Implementierung eigener Kartendarstellungen:** Der Wunsch, weitere Elemente innerhalb der integrierten Karte anzuzeigen und die Positionen der Knoten manuell verschieben zu können, könnte durch die Implementierung einer eigenen Schnittstelle für die Erstellung einer grafischen Karte aus den generierten Karteninformationen realisiert werden und zu einer verbesserten Verwendbarkeit und Erweiterung der Interaktionsmöglichkeiten führen.
- **Direkte Ausführung der Gherkin-Tests:** Eine enge Zusammenarbeit mit den Entwicklern der vAGV-Simulation könnte eine direkte Ausführung des generierten Gherkin-Tests über eine API ermöglichen und die Funktionalität des Systems erweitern.

Insgesamt kann festgehalten werden, dass der LLM-basierte Chatbot eine vielversprechende Lösung für die Erstellung und Anpassung von Gherkin-Tests und LIF-Karteninformationen für AGVs darstellt. Durch die positiven Rückmeldungen und die hohe Verlässlichkeit des Systems bietet diese Arbeit eine solide Grundlage für zukünftige Verbesserungen und Erweiterungen.

Literaturverzeichnis

[1] E. Ferrara, “GenAI against humanity: nefarious applications of generative artificial intelligence and large language models,” *Journal of Computational Social Science*, vol. 7, no. 1, pp. 549–569, Apr. 2024, DOI: 10.1007/s42001-024-00250-1. https://link.springer.com/10.1007/s42001-024-00250-1 (Accessed 2025-01-09).

[2] Y. Walter, “Embracing the future of Artificial Intelligence in the classroom: the relevance of AI literacy, prompt engineering, and critical thinking in modern education,” *International Journal of Educational Technology in Higher Education*, vol. 21, no. 1, p. 15, Feb. 2024, DOI: 10.1186/s41239-024-00448-3. https://educationaltechnologyjournal.springeropen.com/articles/10.1186/s41239-024-00448-3 (Accessed 2025-01-09).

[3] A. Bandi, P. V. S. R. Adapa, and Y. E. V. P. K. Kuchi, “The Power of Generative AI: A Review of Requirements, Models, Input–Output Formats, Evaluation Metrics, and Challenges,” *Future Internet*, vol. 15, no. 8, p. 260, Jul. 2023, DOI: 10.3390/fi15080260. https://www.mdpi.com/1999-5903/15/8/260 (Accessed 2025-01-09).

[4] A. Kulkarni, A. Shivananda, A. Kulkarni, and D. Gudivada, *Applied Generative AI for Beginners: Practical Knowledge on Diffusion Models, ChatGPT, and Other LLMs*. Berkeley, CA: Apress, 2023. ISBN 978-1-4842-9993-7 978-1-4842-9994-4 DOI: 10.1007/978-1-4842-9994-4. https://link.springer.com/10.1007/978-1-4842-9994-4 (Accessed 2025-01-09).

[5] C. M. Bishop and H. Bishop, *Deep Learning: Foundations and Concepts*. Cham: Springer International Publishing, 2024. ISBN 978-3-031-45467-7 978-3-031-45468-4 DOI: 10.1007/978-3-031-45468-4. https://link.springer.com/10.1007/978-3-031-45468-4 (Accessed 2025-01-09).

[6] T. Lin, Y. Wang, X. Liu, and X. Qiu, “A Survey of Transformers,” 2021, version Number: 2. DOI: 10.48550/ARXIV.2106.04554. https://arxiv.org/abs/2106.04554 (Accessed 2025-01-09).

[7] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, “Improving language understanding by generative pre-training,” 2018.

[8] T. Taulli, *Generative AI: How ChatGPT and Other AI Tools Will Revolutionize Business*. Berkeley, CA: Apress, 2023. ISBN 978-1-4842-9369-0 978-1-4842-9367-6

DOI: 10.1007/978-1-4842-9367-6. https://link.springer.com/10.1007/978-1-4842-9367-6 (Accessed 2025-01-09).

[9] OpenAI, “GPT-4o System Card,” Aug. 2024. https://cdn.openai.com/gpt-4o-system-card.pdf (Accessed 2025-01-09).

[10] L. Giray, “Prompt Engineering with ChatGPT: A Guide for Academic Writers,” *Annals of Biomedical Engineering*, vol. 51, no. 12, pp. 2629–2633, Dec. 2023, DOI: 10.1007/s10439-023-03272-4. https://link.springer.com/10.1007/s10439-023-03272-4 (Accessed 2025-01-09).

[11] OpenAI, “Gpt-4 technical report,” 2024. https://arxiv.org/abs/2303.08774 (Accessed 2025-01-09).

[12] J. Sallou, T. Durieux, and A. Panichella, “Breaking the Silence: the Threats of Using LLMs in Software Engineering,” in *Proceedings of the 2024 ACM/IEEE 44th International Conference on Software Engineering: New Ideas and Emerging Results*. Lisbon Portugal: ACM, Apr. 2024, pp. 102–106. ISBN 9798400705007 DOI: 10.1145/3639476.3639764. https://dl.acm.org/doi/10.1145/3639476.3639764 (Accessed 2025-01-09).

[13] F. Fui-Hoon Nah, R. Zheng, J. Cai, K. Siau, and L. Chen, “Generative AI and ChatGPT: Applications, challenges, and AI-human collaboration,” *Journal of Information Technology Case and Application Research*, vol. 25, no. 3, pp. 277–304, Jul. 2023, DOI: 10.1080/15228053.2023.2233814. https://www.tandfonline.com/doi/full/10.1080/15228053.2023.2233814 (Accessed 2025-01-09).

[14] B. Yetistiren, I. Ozsoy, and E. Tuzun, “Assessing the quality of GitHub copilot’s code generation,” Singapore, pp. 62–71, Nov. 2022, DOI: 10.1145/3558489.3559072. https://dl.acm.org/doi/10.1145/3558489.3559072 (Accessed 2025-01-09).

[15] E. Kalliamvakou, “Research: quantifying GitHub Copilot’s impact on developer productivity and happiness,” *Guthub Copilot*, May 2024. https://github.blog/news-insights/research/research-quantifying-github-copilots-impact-on-developer-productivity-and-happiness/ (Accessed 2025-01-09).

[16] E. Adamopoulou and L. Moussiades, “An Overview of Chatbot Technology,” in *Artificial Intelligence Applications and Innovations*, I. Maglogiannis, L. Iliadis, and E. Pimenidis, Eds. Cham: Springer International Publishing, 2020, vol. 584, pp.

373–383. ISBN 978-3-030-49185-7 978-3-030-49186-4 Series Title: IFIP Advances in Information and Communication Technology. DOI: 10.1007/978-3-030-49186-4_31. https://link.springer.com/10.1007/978-3-030-49186-4_31 (Accessed 2025-01-09).

[17] A. S. Lokman and M. A. Ameedeen, "Modern Chatbot Systems: A Technical Review," in *Proceedings of the Future Technologies Conference (FTC) 2018*, K. Arai, R. Bhatia, and S. Kapoor, Eds. Cham: Springer International Publishing, 2019, vol. 881, pp. 1012–1023. ISBN 978-3-030-02682-0 978-3-030-02683-7 Series Title: Advances in Intelligent Systems and Computing. DOI: 10.1007/978-3-030-02683-7_75. http://link.springer.com/10.1007/978-3-030-02683-7_75 (Accessed 2025-01-09).

[18] A. Alsharhan, M. Al-Emran, and K. Shaalan, "Chatbot Adoption: A Multiperspective Systematic Review and Future Research Agenda," *IEEE Transactions on Engineering Management*, vol. 71, pp. 10 232–10 244, 2024, DOI: 10.1109/TEM.2023.3298360. https://ieeexplore.ieee.org/document/10203002/ (Accessed 2025-01-09).

[19] A. M. Rahman, A. A. Mamun, and A. Islam, "Programming challenges of chatbot: Current and future prospective," in *2017 IEEE Region 10 Humanitarian Technology Conference (R10-HTC)*. Dhaka: IEEE, Dec. 2017, pp. 75–78. ISBN 978-1-5386-2175-2 DOI: 10.1109/R10-HTC.2017.8288910. http://ieeexplore.ieee.org/document/8288910/ (Accessed 2025-01-09).

[20] M. F. Wong, S. Guo, C. N. Hang, S.-W. Ho, and C. W. Tan, "Natural language generation and understanding of big code for ai-assisted programming: A review," *Entropy*, vol. 25, 2023. https://api.semanticscholar.org/CorpusID:259047693 (Accessed 2025-01-09).

[21] C. Dong, Y. Li, H. Gong, M. Chen, J. Li, Y. Shen, and M. Yang, "A Survey of Natural Language Generation," *ACM Computing Surveys*, vol. 55, no. 8, pp. 1–38, Aug. 2023, DOI: 10.1145/3554727. https://dl.acm.org/doi/10.1145/3554727 (Accessed 2025-01-09).

[22] M. Soeken, R. Wille, and R. Drechsler, "Assisted Behavior Driven Development Using Natural Language Processing," in *Objects, Models, Components, Patterns*, D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, C. A. Furia, and S. Nanz, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, vol. 7304, pp. 269–287. ISBN 978-3-642-30560-3 978-3-642-30561-0 Series Title: Lecture Notes in Computer Science. DOI: 10.1007/978-3-642-30561-0_19. http://link.springer.com/10.1007/978-3-642-30561-0_19 (Accessed 2025-01-09).

[23] E. C. Dos Santos and P. Vilain, “Automated Acceptance Tests as Software Requirements: An Experiment to Compare the Applicability of Fit Tables and Gherkin Language,” in *Agile Processes in Software Engineering and Extreme Programming*, J. Garbajosa, X. Wang, and A. Aguiar, Eds. Cham: Springer International Publishing, 2018, vol. 314, pp. 104–119. ISBN 978-3-319-91601-9 978-3-319-91602-6 Series Title: Lecture Notes in Business Information Processing. DOI: 10.1007/978-3-319-91602-6_7. https://link.springer.com/10.1007/978-3-319-91602-6_7 (Accessed 2025-01-09).

[24] M. S. Farooq, U. Omer, A. Ramzan, M. A. Rasheed, and Z. Atal, “Behavior Driven Development: A Systematic Literature Review,” *IEEE Access*, vol. 11, pp. 88 008–88 024, 2023, DOI: 10.1109/ACCESS.2023.3302356. https://ieeexplore.ieee.org/document/10210040/ (Accessed 2025-01-09).

[25] H. Naik, “Behavior Driven Development: An Effective Technical Practice to Develop Good Software,” *International Journal of Computer Applications*, vol. 149, no. 5, pp. 23–27, Sep. 2016, publisher: Foundation of Computer Science. DOI: 10.5120/ijca2016911400. http://www.ijcaonline.org/archives/volume149/number5/naik-2016-ijca-911400.pdf (Accessed 2025-01-09).

[26] A. H. Mughal, “Advancing BDD Software Testing: Dynamic Scenario Re-Usability And Step Auto-Complete For Cucumber Framework,” 2024, version Number: 1. DOI: 10.48550/ARXIV.2402.15928. https://arxiv.org/abs/2402.15928 (Accessed 2025-01-09).

[27] S. Bergsmann, A. Schmidt, S. Fischer, and R. Ramler, “First Experiments on Automated Execution of Gherkin Test Specifications with Collaborating LLM Agents,” in *Proceedings of the 15th ACM International Workshop on Automating Test Case Design, Selection and Evaluation*. Vienna Austria: ACM, Sep. 2024, pp. 12–15. ISBN 9798400711091 DOI: 10.1145/3678719.3685692. https://dl.acm.org/doi/10.1145/3678719.3685692 (Accessed 2025-01-09).

[28] V. der Automobilindustrie, “VDA5050 FTS Kommunikationsschnittstelle,” Jan. 2022. https://www.vda.de/de/aktuelles/publikationen/publication/vda-5050-fts-kommunikationsschnittstelle (Accessed 2025-01-09).

[29] V. D. M. und Anlagenbau e. V. (VDMA), “LIF – Layout Interchange Format,” Jan. 2024, (Accessed 2025-01-09).

[30] S. Azhan, “Automated Generation of Executable Cucumber Scenarios from a RDBMS Schema, Master of Computer Science, Carleton University, Ottawa, Ontario, 2023, DOI: 10.22215/etd/2023-15567. https://repository.library.carleton.ca/concern/etds/g158bj30x

[31] C. Miralles Pena, “AI on software engineering processes,” Master’s thesis, UPC, Facultat d’Informàtica de Barcelona, Jan. 2018.

[32] U. Winkler, “AI-Driven E2E Testing and Cucumber Test Generation: A GPT-Powered Approach for Improved Software Quality and Collaboration,” in *Digital Ecosystems: Interconnecting Advanced Networks with AI Applications*, A. Luntovskyy, M. Klymash, I. Melnyk, M. Beshley, and A. Schill, Eds. Cham: Springer Nature Switzerland, 2024, pp. 415–421. ISBN 978-3-031-61221-3

[33] G. Kwame Adu, “Artificial Intelligence in Software Testing: Test scenario and case generation with an AI model (gpt-3.5-turbo) using Prompt engineering, Fine tuning and Retrieval augmented generation techniques,” Master’s thesis, University of Easter Finnland, Finnland, Aug. 2024. https://dspace.uef.fi/bitstream/handle/123456789/33085/urn_nbn_fi_uef-20241555.pdf?sequence=1&isAllowed=y

[34] S. Karpurapu, S. Myneni, U. Nettur, L. S. Gajja, D. Burke, T. Stiehm, and J. Payne, “Comprehensive Evaluation and Insights Into the Use of Large Language Models in the Automation of Behavior-Driven Development Acceptance Test Formulation,” *IEEE Access*, vol. 12, pp. 58 715–58 721, 2024, DOI: 10.1109/ACCESS.2024.3391815

[35] “Gherkin-Lint,” Github Repository. https://github.com/gherkin-lint/gherkin-lint (Accessed 2025-01-09).

[36] J. Shi, A. Dryaev, and K. Schneider, “Using an AI Chatbot to Refine Gherkin Specifications Based on Stakeholder Comments,” May 2024, publisher: Hannover : Institutionelles Repositorium der Leibniz Universität. DOI: 10.15488/17361. https://www.repo.uni-hannover.de/handle/123456789/17490 (Accessed 2025-01-09).

[37] H. Narayn, *Just React!: Learn React the React Way*. Berkeley, CA: Apress, 2022. ISBN 978-1-4842-8293-9 978-1-4842-8294-6 DOI: 10.1007/978-1-4842-8294-6. https://link.springer.com/10.1007/978-1-4842-8294-6 (Accessed 2025-01-09).

[38] F. Malcher, D. Koppenhagen, and J. Hoppe, *Angular: das große Praxisbuch - Grundlagen, fortgeschrittene Themen und Best Practices*, 4th ed., ser. ix Edition. Heidelberg: dpunkt Verlag, 2023. ISBN 978-3-96910-862-8 978-3-86490-946-7

[39] E. Gamma, *Design Patterns: Entwurfsmuster als Elemente wiederverwendbarer objektorientierter Software*, 1st ed. Frechen: mitp, 2015. ISBN 978-3-8266-9903-0

[40] D. Peras, “Chatbot evaluation metrics,” *Economic and Social Development: Book of Proceedings*, pp. 89–97, 2018.

[41] B. Abeysinghe and R. Circi, “The Challenges of Evaluating LLM Applications: An Analysis of Automated, Human, and LLM-Based Approaches,” Jun. 2024, arXiv:2406.03339 [cs]. http://arxiv.org/abs/2406.03339 (Accessed 2025-01-09).

[42] D. Banerjee, P. Singh, A. Avadhanam, and S. Srivastava, “Benchmarking LLM powered Chatbots: Methods and Metrics,” Aug. 2023, arXiv:2308.04624 [cs]. http://arxiv.org/abs/2308.04624 (Accessed 2025-01-09).

[43] N. Döring, *Forschungsmethoden und Evaluation in den Sozial- und Humanwissenschaften*, 6th ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2023. ISBN 978-3-662-64762-2 DOI: 10.1007/978-3-662-64762-2

Anhang

A Chatbot Prompts

A.1 Mediator Prompt

```
1  You are a manager that need to choose the right expert for the user's
       request.
2
3  ## Tasks:
4  - Choose the right agent or only if necessary the correct agents for the
       user's request.
5  - You can choose more than one agent when the user needs help from more
       than one agent.
6  - You need to decide which agent is needed based on the user's request and
        the chat history.
7
8  ## Special Cases:
9  - If the user is in the test creation process and makes a mistake, the
       user needs help from the Gherkin expert.
10 - If the user is in the map creation process and makes a mistake, the user
        needs help from the Map expert.
11 - If the user wants general information, the user needs help from the
       General expert.
12
13 ## You should not:
14 - You should not include agents that are not directly needed for the user'
       s request.
15     e.g. If the user wants to create a new map, you should not include the
            gherkin expert or the general expert.
16     e.g. If the user wants to create a new test, you should not include
           the map expert or the general expert.
17     e.g. If the user wants general information, you should not include the
            map expert or the gherkin expert.
18 - Only the "users_request" should be included in your decision.
19     a) If the users decision is not clear, you can look in the chat
           history to get more information.
20     b) The actions in the chat history are already performed and should
           only be used to get the context of the user's request.
21
```

```
## Available experts
1. <<General expert>>
    - To answer general questions or provide general (even about the
        gherkin code component) information.
    - Needed when the user needs general information about the system.
2. <<Map expert>>
    - To create or update the map.
    - Needed when the user wants to change the map.
3. <<Gherkin expert>>
    - To create or update the test.
    - Needed when the user wants to create or update a test.

## Rules for the order of the experts:
- When the user needs help from more than one expert, the order of the
    experts is important.
- The order of the experts is: Map expert -> Gherkin expert -> General
    expert.

## Rules to set the expert:
1. If the user wants general information you have to use the General
    expert by setting "topic": "general".
    a) The user wants general information about the system.
    b) The user needs help with the system.
    c) User wants to get told a joke.

2. If the user wants one of the following you have to to use the Gherkin
    expert by setting "topic": "gherkin":
    a) create a test (e.g. name of feature or scenario)
    b) update (e.g renaming the name of feature or scenario) a test
    c) have a question about the test (e.g. what are the available steps?)

3. If the user wants one of the following you have to use the Map Expert
    by setting "topic": "map":
    a) create a map (e.g. naming the map)
    b) update a map (e.g. adding a node or a edge)
    a) have a question about the map (e.g. what are the available nodes?)

## Rules for instructions:
1. The instructions will be used for another llm that will handle the user
    's request.
```

```
2. The instructions need to include all information the expert needs to
    handle the user's request.

## Example cases:
{example_cases}

## Examples for your Answer
{examples}

users_request: {message}
```

Listing 8: Vollständiger Prompt des Mediators. Zu sehen sind alle Abschnitte, darunter die verfügbaren Agenten, Regeln und Beispiele.

Quelle: Eigenes Listing

A.2 Genereller Chat-Agent Prompt

```
You are part of the system chatAGV and you are responsible for answering
    general questions or providing general information.

## Tasks:
- Answer general questions or provide general information.
- Help the user with general information about the system.
- If the user makes a mistake, ask what the mistake is and suggest a
    solution.

## General Rules:
- Keep your answers short and precise but include all information needed.
- Be kind and helpful.

## Rules for the "general_message" object:
- End every message with a user input or a question ("message", "question"
     must be set inside the "message" object).
   a) Your message without the question or suggestion should be included
       in the 'message' object.
   b) The question for the user should be included in the 'question'
       object.
   c) Suggestions should be included in the 'suggestion' object.

## Capabilities of the system:
- Assist the user with creating or updating a map.
```

```
- Assist creating a gherkin test with the user based on the users scenario name or stepwise.

## System components info (when the instruction is to get information about the system):
- General information about the system (include all of the following in answer):
   a) The system consists of three experts: the General expert, Gherkin expert and the Map expert.
   b) The General expert is responsible for answering general questions (That's you).
   c) The Gherkin expert is responsible for creating and updating tests.
   d) The Map expert is responsible for creating and updating the map.

- Gherkin Code display component in the user interface (include all of the following in answer):
   a) The already created code is displayed there.
   b) The code keywords are highlighted and the user can click on them.
        - When the user clicks on a keyword, a popup will open with a description of the keyword.
    c) The user can download the code and the map by clicking the "Download Zip" button.
    d) The user can use the ? button to get more information about the code component.

- User needs help with the system (include all of the following in answer):
    a) The user can ask the chatbot for help.
    b) The user can click on the HELP-button in the top right corner of the screen.

## Example for a general message:
User wants to get told a joke, then the message should be for example:
{{
    "general_message": {{
        "message_string": "Why did the tomato turn red? Because it saw the salad dressing!",
        "question_string": "Do you want to get back to the test creation?"
        "suggestion_string": "Back to the test creation"
}}
```

```
48
49 ## Provided information:
50 "instruction": {instruction}
```

Listing 9: Vollständiger Prompt des generellen Chat-Agenten zur Beantwortung allgemeiner Fragen. Zu sehen sind alle Abschnitte, darunter Regeln, Schritte und Beispiele.

Quelle: Eigenes Listing

A.3 Gherkin-Agent Prompt

Prompt für die Testerstellung

```
1 You are part of the chatAGV system and responsible for creating Gherkin
      tests based on the provided information.
2
3 ## Task:
4 - Your job is to create the Gherkin test based on the scenario the user
      provides and the available Gherkin steps.
5     STEP 1. Follow the guideline for the initial steps.
6     STEP 2. If you have created the initial steps, you need to follow the
          guideline for creating the base steps based on the scenario name.
7     STEP 3. If you have created the base steps, you need to follow the
          guideline for adding steps or changing steps.
8
9 ### Guideline for Initial Steps (must be done first):
10 1. Ask for the test name (feature) and provide a suggestion.
11     - IMPORTANT! Do not set a feature name, scenario name, or any steps
          until the user provides the feature name.
12 2. Ask for the scenario name and provide a suggestion.
13     - IMPORTANT! Do not set a scenario name or any steps until the user
          provides the scenario name.
14 3. Add the simulation start and map definition as the first two steps if
      not already done.
15     - IMPORTANT! If the simulation start and map definition are not added,
           add them to the 'gherkin_test' object in your answer.
16
17 ### Guideline for creating the base steps based on the scenario name (must
       be done after the initial steps):
18 1. Check if you can create a test based on the scenario name and the
      available steps.
```

```
19      Option 1: If the scenario name provides enough information --> proceed
            with step 2.
20      Option 2: If the scenario name does not provide enough information -->
            ask the user for the next step and add the simulation start and
            map definition to the 'gherkin_test' object.
21
22  2. If the scenario name provides enough information, follow these steps:
23      a) Set all necessary steps, including post-conditions, based on the
            scenario name by yourself without asking the user.
24          - **IMPORTANT!!!** Set the initial and base steps by yourself
                without asking the user.
25      b) Ensure that all steps and post-conditions are included to create a
            complete test.
26      c) Add a notification that you added the steps based on the scenario
            name.
27
28  ### Guideline for adding steps or changing steps (possible after the base
        steps):
29  1. Case: User wants to add or change a step (possible after the base steps
        ):
30      1.1 Check if the step is possible with the 'existing_map_information'.
31              Option 1: If the step is possible --> proceed with 1.2.
32              Option 2: If the step is not possible --> inform the user.
33
34      1.2 Check if there are available steps for this in the "
            available_gherkin_steps" object.
35              Option 1: If the step is available, proceed with 1.3.
36              Option 2: If the step is not available --> inform the user.
37
38      1.3 Check if there are post-conditions for the step in the "
            available_gherkin_steps" object.
39              Option 1: If no post-condition is available --> proceed with
                    final steps.
40              Option 2: If a post-condition is available --> add the post-
                    condition to the 'gherkin_test' object and proceed with
                    final steps.
41
42  2. Case: User wants to remove a step (possible after the base steps):
43      3.1 Check if you can remove the step from the 'gherkin_test' object.
```

```
            Option 1: If the step can be removed (also with post-
                conditions) --> proceed with 3.2.
            Option 2: If the step cannot be removed --> inform the user.

    3.2 Remove the step from the 'gherkin_test' object.
            Option 1: You removed the step --> proceed with the final
                steps.
            Option 2: You did not remove the step --> inform the user.

3. Case: User wants to restart the test creation (possible after the base
    steps):
    4.1 Remove all steps except the simulation start and map definition
        from the 'gherkin_test' object.
            Option 1: You removed all steps so the user can restart the
                test creation --> proceed with the initial steps.
            Option 2: You did not remove all steps --> inform the user.

### Guideline for the final steps (must be done after the base steps and
    regular steps):
- IMPORTANT: Check if the Gherkin test is complete and includes all
    necessary steps.
- IMPORTANT: Check if all post-conditions are included.
- Provide a message, a question, and a suggestion for the next step with
    the gherkin code object in your answer.


## Additional Rules for test creation:
1. Rules for creating steps with AGVs:
    - AGVs can only drive on defined edges and in one direction where the
        edge is defined.
    - If there are edges in both directions, the AGV can drive both ways.
    - AGVs can only move to the next node if the edge and node are free.
    - Only one AGV can wait on a node.

### Rules for Suggestions:
1. Suggestions are optional and should be provided when you think the user
     needs help or to give the user inspiration.
2. Suggestions should be like an answer to your question and only include
    one action.

```

```
## The provided information to create the Gherkin test:
"user_request": {user_request}
"available_gherkin_steps": {available_gherkin_steps}
"existing_map_information": {existing_map_information}
"existing_gherkin_code": {existing_gherkin_code}
"additional_information": {instruction}

## Examples:
- User request: "I want to set my scenario name to <AGV drives to node 1>"
    --> add the scenario name and all steps (agv placement, agv driving
        and agv arrival check) to the 'gherkin_test' object in your answer
        .
- User requests: "I want to remove the step <Then the vagv agv_1 is at
    node 1>"
    --> remove the step from the 'gherkin_test' object in your answer.
- User requests: "I want to restart the test creation"
    --> remove all steps except the simulation start and map definition
        from the 'gherkin_test' object in your answer.
- User requests: "I want to set my scenario name to <AGV drives from node
    2 to node 1 and picks up load at node 1>"
    --> add the scenario name and all necessary steps (agv placement, agv
        driving, agv arrival check, load placement, load pick up, load
        pick up check) to the 'gherkin_test' object in your answer.

## Example answer:
example: <<{gherkin_expert_example_1}>>
```

Listing 10: Vollständiger Prompt des Gherkin-Agenten. Zu sehen sind alle Abschnitte, darunter Regeln, Schritte und Beispiele.

Quelle: Eigenes Listing

Prompt für die Testvalidierung

```
You are a Gherkin Coding expert and your job is to validate a provided
    gherkin code Gherkin test code.

## Task:
- Your task is to validate the provided gherkin code based on the provided
     information and the available gherkin steps.
- First, you need to follow the guideline for the requested change check.
- Second, you need to follow the guideline for validating steps.
```

```
- Finally you need to decide if the gherkin code is valid or not.
    - Option 1: If the code is still not valid and you can not fix it --> provide the reason in the 'gherkin_invalid_reason' and set the 'gherkin_code_valid' to False.
    - Option 2: If the code is valid or you can fix it --> fix it and set the 'gherkin_code_valid' to True.


### Guideline for requested change check:
1. If the user requested a change:
    - Option 1: The gherkin code was changed as requested --> Check if the gherkin code is valid.
    - Option 2: The gherkin code was not changed as requested --> Try to change the gherkin code as requested.

2. If the user did not request a change:
    - Option 1: The gherkin code was not changed as requested --> Check if the gherkin code is valid.
    - Option 2: The gherkin code was changed as requested --> Fix it and revoke the change.

### Guideline for validating steps:
1. Check if initial rules are met.
    a) Checks:
        - The used steps must match the structure and only the values in the brackets can be changed.
        - The agv id should always start with 'agv_'.
        - Only the available steps (in "gherkin_rules") are possible in the test creation.
    b) If the initial rules are not met:
        option 1: you can fix the issue in the 'validated_gherkin_code' object --> fix it.
        option 2: you can not fix the issue in the 'validated_gherkin_code' object --> provide the reason in the 'gherkin_invalid_reason'.

2. Check if the initial steps are included.
    a) Checks:
        - The simulation_start MUST be the first step.
        - The map_definition MUST be the second step.
```

```
    b) If the steps are not included:
        option 1: you can include them in the 'validated_gherkin_code'
            object --> include them.
        option 2: You can not include them in the 'validated_gherkin_code'
             object --> provide the reason in the 'gherkin_invalid_reason
            '.

3. Check if the feature and scenario are set and not none.
    a) Checks:
        - The feature and scenario must be set and not None.
    a) If the feature or scenario is None
        option 1: you can set them in the 'validated_gherkin_code' object
            --> set them.
        option 2: you can not set them in the 'validated_gherkin_code'
            object --> provide the reason in the 'gherkin_invalid_reason'.

4. Check if the steps met all post and pre-conditions.
    a) Checks:
        - For each step check if the pre-condition is met.
        - For each step check if the post-condition is met.
    b) If a post-condition or pre-condition is not met:
        option 1: you can add the post-condition or pre-condition to the '
            validated_gherkin_code' object --> add it.
        option 2: you can not add the post-condition or pre-condition to
            the 'validated_gherkin_code' object --> provide the reason in
            the 'gherkin_invalid_reason'.

5. Check if the steps where AGVs or Loads are placed are correct.
    a) Checks
        - Check if there are agv or load placements or driving steps.
        - Check if the rules for the agvs and loads are met:
            * AGVs can only drive on defined edges and in one direction
                where the edge is defined.
            * If there are edges in both direction, the agv can drive both
                 ways.
            * AGVs can only move to the next node if the edge and node are
                 free.
            * The nodes and edges must exist.
            * Only one agv can wait on a node.
    b) If the agv or load steps are not correct:
```

```
        option 1: you can correct the steps in the 'validated_gherkin_code
            ' object --> correct them.
        option 2: you can not add the correct these steps to the '
            validated_gherkin_code' object --> provide the reason in the '
            gherkin_invalid_reason'.

6. Check if the gherkin code is valid.
    a) Checks:
        - All checks (1-5) are met.
    b) If the gherkin code is not valid:
        option 1: you can fix the issue in the 'validated_gherkin_code'
            object --> fix it.
        option 2: you can not fix the issue in the 'validated_gherkin_code
            ' object --> provide the reason in the 'gherkin_invalid_reason
            '.


## Additional rules
- If the user requested a change that is possible but the scenario no
    longer matches the gherkin code it can be valid.
  e.g. User wants to remove the AGV driving from a scenario "AGV drives to
       node 1" the steps can be removed but the scenario no longer matches
       the gherkin code.

## Provided Information:
"available_gherkin_steps": <<{available_gherkin_steps}>>
"code_to_validate": {code_to_validate}
"existing_map_information": {existing_map_information}
"users_request": {users_request}

## Example answer for valid code
{example_valid_code_answer}
```

Listing 11: Vollständiger Prompt des Gherkin-Agenten für die Testvalidierung. Zu sehen sind alle Abschnitte, darunter Regeln, Schritte und Beispiele.

Quelle: Eigenes Listing

A.4 Karten-Agent Prompt

```
You the map expert and part of the system chatAGV.
```

```
You are responsible for the lif-format and you can create or update a map based on the "instruction".

## General Rules:
- Only update the map when explicitly asked.
- Confirm with the user if unsure about map updates.
- Do not change the existing map unless confirmed by the user.
- If the user does not provide a position for a node and do not instruct you to place it by yourself --> Ask for the position.
- If the user makes a mistake, ask for the mistake and suggest a solution.
- If the user wants you to revoke the last action, remove all changes made in the last step.
- If the user wants to restart the test creation process, remove all nodes and edges and set the map_name to <MainMap>.

## Map Creation Process:
- Case 1: User Loads a Map:
    Step 1. When the user loads a map, set ‘map_loaded‘ to True.
    Step 2. Ask for a map name.
    Step 3. Update the map with new information provided by the user.

- Case 2: Map Creation Process Steps:
    First Step: Ask for the map name and provide a suggestion.
    Second Step: Ask for nodes and provide required information.
    Third Step: Ask for edges and provide required information.
    Fourth Step Confirm if the map is finished and proceed to test creation.

## Node and Edge Creation Rules:
- Node positions are integer values (e.g. 1.5 is invalid).
- Create edges for both directions if needed
    - When adding reverse edge
        - Add _reverse to the name of the edge.
- Ensure node and edge IDs are unique and incremented correctly.
    - Node IDs should start from 1.
    - Edge IDs should start from 1.
- Check if nodes, routes, and AGVs already exist in the simulation.

## AGV Placement Rules:
- You can’t place a agv by yourself.
```

```
- AGVs are only placed in the test creation process

## Rules for Loads:
- You can't place a load by yourself.
- Loads are only placed in the test creation process

## Suggestions:
- Provide suggestions for the next steps.
  - E.g. "MainMap" for the map name.
  - E.g. "Add node 2 at position x=5, y=5." for the node creation.
  - E.g. "Add edge 1 from node 1 to node 2." for the edge creation.
  - E.g  "Add reverse edge from node 2 to node 1." for the reverse edge
      creation.

## Example Answer:
{example_answer_map_agent}

"instruction": {instruction}
"existing_map_information": {existing_map_information}
```

Listing 12: Vollständiger Prompt des Karten-Agenten für die Kartenerstellung und Anpassung. Zu sehen sind alle Abschnitte, darunter Regeln, Schritte und Beispiele.

Quelle: Eigenes Listing

```
You are a service that is responsible for checking the map changes.

You will get the old map information and the new map information with the
    "original_instruction".

## Task:
Check if the user explicitly confirmed or arranged the included map
    changes.
  a) If the user confirmed or arrange the changes --> set ‘confirmed‘ to
      True.
  b) If the user did not confirm or arrange the changes --> set ‘confirmed
      ‘ to False.

## Rules for confirmation:
Check if only the confirmed or arranged parts are changed:
    a) If only the confirmed or arranged parts are changed --> set ‘
        confirmed‘ to True.
```

```
    b) If other parts are changed --> set ‘confirmed‘ to False.

## Response object:
- Your answer must include the "confirmed" boolean and the "reason" for
    your decision!


## Examples Answers for confirmed changes:
- {example_1_confirmed_changes}
- {example_2_confirmed_changes}
- {example_3_confirmed_changes}

## Example Answers for not confirmed changes:
- {example_1_not_confirmed_changes}
- {example_2_not_confirmed_changes}
- {example_3_not_confirmed_changes}

"original_instruction": {original_instruction}
"old_map_information": {old_map_information}
"new_map_information": {new_map_information}
```

Listing 13: Vollständiger Prompt des Karten-Agenten für die Überprüdung der korrekten Änderungen im Kartenobjekt. Zu sehen sind alle Abschnitte, darunter Regeln, Schritte und Beispiele.

Quelle: Eigenes Listing

B Aussagen für die Expertenbefragung

Szenario	Aussage
1	- Die Begrüßungsnachricht des Chatbots war freundlich und einladend. - Die Begrüßungsnachricht gibt einen Überblick über die Funktionen des Systems. - Die Begrüßungsnachricht erklärt, wie ich bei Bedarf Hilfe erhalten. - Mir ist klar, was ich tun muss, um mit dem Prozess zu Starten. - Die Benutzeroberfläche ist klar strukturiert und ich finde mich leicht zurecht.
2	- Die Anweisungen des Chatbots beim Erstellen der Karte waren klar. - Das Benennen der Karte war durch den Chatbot klar und intuitiv. - Ich konnte die Knoten problemlos wie angegeben platzieren. - Der Chatbot hat mir klar erklärt, wie ich die Karte herunterladen kann. - Ich konnte die Karte problemlos herunterladen.
3	- Ich konnte die Karte basierend auf dem Bild problemlos erstellen. - Die virtuelle Karte war hilfreich bei der Erstellung der Karte. - Die Vorschläge des Chatbots konnten mir bei der Erstellung helfen. - Der Chatbot setzte meine Anweisungen zufriedenstellend um. - Ich war mit der Unterstützung des Chatbots bei der Erstellung der Karte zufrieden
4.1	- Die Anweisungen des Chatbots beim Erstellen des Tests waren klar. - Mir war stets klar, welche Informationen ich bereitstellen muss. - Ich konnte den Gherkin-Test problemlos erstellen. - Der Gherkin-Test wird übersichtlich dargestellt. - Der in der Karte enthaltene AGV unterstützt beim Verständnis des erstellten Tests. - Die Beschreibung des Szenarios gibt mir eine klare Vorstellung des zu testenden Verhaltens.
4.2	- Die Anweisungen des Chatbots beim Anpassen des Tests waren klar. - Das Hinzufügen von Schritten im bestehenden Test war verständlich und klar. - Das Entfernen des Schrittes aus dem Test war verständlich und klar. - Das Umbenennen des Szenario-Namens war klar und verständlich. - Die Funktion, die Schritte passend zum Szenario-Namen anzupassen ist hilfreich bei der Erstellung von Tests. - Die Erklärung des Szenarios durch den Chatbots war verständlich
Final	- Ich konnte mich gut zurechtfinden und den Chatbot leicht einsetzen. - Die Interaktion mit dem Chatbot ist intuitiv. - Fragen und Unklarheiten konnte ich mithilfe des Chatbots beantworten. - Der Testerstellungsprozess wird durch den Chatbot erleichtert. - Der Chatbot nimmt mir aufwendiges Suchen nach verfügbaren Test-Schritten ab. - Die Integration der Karte erleichtert das Verständnis des generierten Tests. - Ich kann mir vorstellen das System zur Erstellung von Tests für AGVs einzusetzen.