

MASTERARBEIT
Master of Science (M.Sc.) Digital Reality

Unsicherheitsquantifizierung von Large Language Models bei der Automatisierung und Steuerung interaktiver Webprozesse

Hamburg, den 7. Mai 2026
Hendrik Pietrzyk

Erstprüferin: Prof. Dr. Larissa Putzar
Zweitprüfer: Dipl. Inf. Rüdiger Höfert

HOCHSCHULE FÜR ANGEWANDTE
WISSENSCHAFTEN HAMBURG
Fakultät Informatik und Digitale Gesellschaft
Finkenau 35
22081 Hamburg

Zusammenfassung

Die Arbeit untersucht die Unsicherheitsquantifizierung von Large Language Models (LLMs) bei der Automatisierung und Steuerung von interaktiven Webprozessen. Ziel ist es, zu analysieren, wie zuverlässig Sprachmodelle bei der Planung und Ausführung von Aktionen in einer webbasierten Prozessmanagementanwendung arbeiten und ob sich die Unsicherheit währenddessen messbar machen lässt. Hierzu wurde am Beispiel der Anwendung TSC-Online ein System entwickelt, welches Nutzereingaben in strukturierte Aktionssequenzen übersetzt und diese automatisiert im Browser ausführt. Dazu wurden drei Ansätze zur Unsicherheitsbewertung evaluiert: tokenwahrscheinlichkeitsbasierte Metriken, verbalisierte Selbstangaben der eigenen Antwortkonfidenz sowie ein konsistenzbasiertes Verfahren, das mehrere generierte Antworten auf semantische Ähnlichkeit überprüft. Die Evaluation erfolgte anhand realistischer Interaktionsszenarien mit Referenzlösungen. Die Ergebnisse zeigen, dass alle untersuchten Metriken grundsätzlich zur Erkennung von Modellunsicherheit geeignet sind. Am zuverlässigsten erwies sich jedoch die konsistenzbasierte Metrik. Insgesamt leistet die Arbeit einen Beitrag zur sicheren Nutzung von LLMs in produktiven, praxisnahen Anwendungsprozessen und KI-Systemen.

Abstract

This thesis investigates the uncertainty quantification of large language models (LLMs) in the automation and control of interactive web processes. The aim is to analyze how reliably LLMs perform when planning and executing actions in a web-based process management application, and whether their uncertainty can be measured during this process. Using TSC-Online as an example application, a system was developed that translates user inputs into structured action sequences and executes them automatically in the browser. To this end, three approaches to uncertainty assessment were evaluated: token-probability-based metrics, verbalized self-reports of response confidence, and a consistency-based method that checks multiple generated responses for semantic similarity. The evaluation was conducted using realistic interaction scenarios with reference solutions. The results show that all investigated metrics are generally suitable for detecting model uncertainty. However, the consistency-based metric proved to be the most reliable. Overall, the thesis contributes to the safe use of LLMs in productive, practice-oriented application processes and AI systems.

Inhaltsverzeichnis

Abbildungsverzeichnis	III
Tabellenverzeichnis	V
1. Einleitung	1
1.1. Zielsetzung der Arbeit	2
1.2. Aufbau der Arbeit	3
1.3. Vorstellung des Unternehmens	3
2. Theoretische Grundlagen	4
2.1. Large Language Models	4
2.1.1. Tokens und Embeddings	4
2.1.2. Transformer-Modelle	6
2.1.3. Textgenerierung	11
2.2. Prompt Engineering	13
2.3. Unsicherheitsquantifizierung von LLMs	15
2.3.1. Unsicherheit und Konfidenz	16
2.3.2. Tokenwahrscheinlichkeitsbasierte Unsicherheitsquantifizierung	17
2.3.3. Verbalisierte Unsicherheitsquantifizierung	19
2.3.4. Konsistenzbasierte Unsicherheitsquantifizierung	20
3. Aufbau und Architektur der Anwendung	23
3.1. Beispielapplikation TSC-Online	23
3.2. Entwicklung und Architektur	25
3.2.1. Funktionale Beschreibung	26
3.2.2. Technische Beschreibung	28
3.2.3. Verwendetes LLM	33
3.2.4. Entwicklungs- und Systemumgebung	33
4. Evaluation	35
4.1. Evaluationsmetrik	35
4.2. Forschungsfrage und Hypothese	35

4.3.	Methodik	37
4.3.1.	Tokenwahrscheinlichkeitsbasierte UQ	37
4.3.2.	Verbalisierte UQ	39
4.3.3.	Konsistenzbasierte UQ	40
4.4.	Versuchsaufbau	41
4.4.1.	Interaktionsszenarien	41
4.4.2.	Kontext Hintergrundwissen	42
4.5.	Implementierung der Datenerhebung	42
4.5.1.	Tokenwahrscheinlichkeiten	43
4.5.2.	Verbalisierte Unsicherheit	45
4.5.3.	Konsistenzbasierte Unsicherheit	46
5.	Ergebnisse	48
5.1.	Zusammengefasste Ergebnisse	48
5.2.	Ergebnisse ohne zusätzliches Hintergrundwissen	50
5.3.	Ergebnisse mit zusätzlichem Hintergrundwissen	51
5.4.	Hypothesenprüfung	53
5.5.	Ergebnisse der Klassifikation	54
6.	Diskussion	56
6.1.	Interpretation der Ergebnisse	56
6.2.	Einfluss von Hintergrundwissen	58
6.3.	Einordnung in den Forschungsstand	58
6.4.	Stärken, Schwächen und Verzerrungen	59
6.5.	Praktische Implikationen	61
6.6.	Ausblick	61
7.	Fazit	64
	Literatur	65
A.	Anhang	70
A.1.	Automatisierungsanwendung	70
A.2.	Hintergrundwissen	72
A.3.	Ground Truth Labels der Interaktionsszenarien	73
A.4.	Deskriptive Statistiken der zusammengefassten Ergebnisse	75

Abbildungsverzeichnis

2.1.	Resultierende Tokens für den Eingabesatz „Der Motor ist defekt.“ bei Verwendung des OpenAI Tokenizers für GPT-4o, Quelle: https://platform.openai.com/tokenizer	5
2.2.	Links: Zeigt den Vektoroffset von drei Wortpaaren anhand des Geschlechts-Features, Rechts: Zeigt die Projektionen zweier Wortpaare anhand ihrer Singular/Plural Relation, Quelle: Mikolov et al., 2013	6
2.3.	Die Berechnung der Self-Attention und damit des finalen Kontextvektors z am Beispiel des zweiten Tokens „journey“, Quelle: Raschka, 2025, S. 69 . . .	9
2.4.	Die Architektur des Transformer-Modells wie es erstmals von Vaswani et al. vorgestellt wurde. Auf der linken Seite befindet sich der Encoder, der die Eingabe verarbeitet und auf der rechten Seite der Decoder, welcher die Ausgabe generiert. Quelle: Vaswani et al., 2017	10
2.5.	Beispiel der autoregressiven Texterzeugung. Beginnend mit der Eingabesequenz „Hello, I am“ wird im ersten Schritt das Token „a“ generiert und der Sequenz angehängen. Die Eingabesequenz im zweiten Durchgang lautet demnach „Hello, I am a“. Diese Schritte werden wiederholt bis die Ausgabesequenz vollständig generiert wurde. Quelle Raschka, 2025, S. 122	12
2.6.	Demonstration von Chain-of-Thought Prompting bei arithmetischen Aufgaben gegenüber Standard Prompting. Reasoning Prozesse wurden farblich hervorgehoben. Quelle: Wei et al., 2023	14
2.7.	In diesem Beispiel der autoregressiven Tokenerzeugung sind die drei Tokenkandidaten mit den höchsten Wahrscheinlichkeiten p_i für die Tokens „model“ und „ready“ an den Positionen $i = 5$ und $i = 6$ zu sehen, welche nach der Greedy Decoding Strategie ausgewählt wurden.	18
3.1.	Ein Ausschnitt von TSC-Online anhand eines Testdatensatzes im Angebotsbereich	24
3.2.	Die DOM-Struktur bis zu dem Navigationselement „Angebote“ von TSC-Online	26

3.3.	Ablaufdiagramm der Webseitensteuerung. Durch eine Nutzereingabe wird die Steuerung initiiert. Interaktive Elemente der Webseite werden extrahiert und zusammen mit der Nutzereingabe und einem Systemprompt an ein LLM gesendet. Dieses antwortet mit einer strukturierten Aktionssequenz. Aktionen werden ausgeführt und die Sequenz gegebenenfalls überarbeitet, bis die Aufgabe als abgeschlossen angesehen wird.	27
3.4.	Sequenzdiagramm der Automatisierungskette. Die Aufgabe wird durch Nutzereingabe initiiert und führt zu automatisch ausgeführten Aktionen im Browser.	29
3.5.	Hintergrund: interaktive Elemente werden zur Veranschaulichung grün hervorgehoben, rote sind deaktivierte oder schreibgeschützte interaktive Elemente. Vordergrund: Der Chatverlauf der Automatisierungsanwendung. Bei jeder Sequenz wird das Reasoning des LLMs als Nachricht eingefügt.	31
3.6.	Struktur der LocatorActionResults-Antwort (JSON) und der enthaltenen ActionResult-Objekte	32
4.1.	Struktur der ChatCompletionTokenLogprob-Antwort von der OpenAI API und der enthaltenen TokenLogprob-Objekte	43
4.2.	Struktur der ActionMetricsResult. Jede Aktion beinhaltet eine Liste von LogProbsSubEntry-Einträgen. Diese enthalten für jeden extrahierten Parameter (in diesem Fall description) die Tokendetails, wie das Token selbst, die Logprobs dieses Tokens sowie die Liste der Toptokens dieser Position.	45
5.1.	ROC-Kurven der zusammengefassten Betrachtung der einzelnen Unsicherheitsmetriken.	50
5.2.	ROC-Kurven der einzelnen Unsicherheitsmetriken ohne Hintergrundwissen.	51
5.3.	ROC-Kurven der einzelnen Unsicherheitsmetriken mit Hintergrundwissen.	52
5.4.	ROC-Kurven der zusammengefassten Betrachtung der einzelnen Unsicherheitsmetriken für die Themenklassifikation.	55

Tabellenverzeichnis

4.1. Operationalisierung der zu untersuchenden Unsicherheit den verschiedenen Dimensionen.	38
4.2. Übersicht der Unsicherheitsmetriken unterteilt in Greybox- und Blackbox-Methoden	41
5.1. AUROC-Werte der untersuchten Unsicherheitsmetriken über alle Test-Szenarien S1 bis S7. Szenarien ohne Hintergrundwissen wurden durch den Index 0, mit Hintergrundwissen mit Index 1 und aggregierte Messwerte ohne Index gekennzeichnet. Die Spalte „Overall“ berücksichtigt die Messwerte aller Szenarien. Jede Reihe beschreibt die AUROC Ergebnisse einer Metrik, wobei verbal die verbalisierten und consistency die konsistenzbasierten Unsicherheitsmetriken bezeichnen. Die Metriken der maximalen und durchschnittlichen Entropie werden durch $\max(H)$ und $\text{avg}(H)$, der geringsten und durchschnittlichen Tokenwahrscheinlichkeiten durch $\min(p)$ und $\text{avg}(p)$ dargestellt. . . .	49
5.2. Messwerte der untersuchten Unsicherheitsmetriken der 10 aufgetretenen inkorrekten Klassifikationen.	55
A.1. Deskriptive Statistik der erfassten, zusammengefassten Konfidenzmesswerte. Die Tabelle zeigt die Anzahl aller Messwerte (count), den durchschnittlichen Konfidenzwert (mean), die Standardabweichung (std), den niedrigsten (min) und höchsten (max) Messwert, sowie die drei verschiedenen Perzentile (25 %, 50 %, 75 %). Insgesamt wurden 5409 Aktionen durchgeführt und erfasst. . . .	75

1. Einleitung

Die stetige Weiterentwicklung großer Sprachmodelle (Large Language Models, LLMs) und deren Integration in Softwaresysteme eröffnet neue Möglichkeiten für die Automatisierung von Prozessen, auch im Bereich der webbasierten Nutzerinteraktion. Durch ihre Fähigkeit, natürliche Sprache zu verstehen und in strukturierte Ausgaben zu überführen, können LLMs beispielsweise dazu eingesetzt werden, Nutzer*innen innerhalb von Anwendungen mit Hilfe zur Seite zu stehen oder Anwendungen selbst zu steuern.

Ein zentrales Problem bei diesem Einsatz ist jedoch die Frage nach der Verlässlichkeit der vom Modell vorgeschlagenen oder sogar durchgeführten Aktionen. Insbesondere in sicherheitskritischen oder geschäftsrelevanten Anwendungen ist es entscheidend, dass das System nicht nur korrekte Anweisungen generiert, sondern auch realistisch einschätzen kann, wie sicher beziehungsweise unsicher es sich bei seinen Vorschlägen ist. Diese Abschätzung der Unsicherheit nennt man Unsicherheitsquantifizierung (Uncertainty Quantification, UQ).

Ziel dieser Arbeit ist es, die Sicherheit von LLMs im Kontext der Steuerung einer konkreten Prozessmanagementanwendung zu untersuchen. Im Fokus steht dabei die Frage, inwiefern sich aus internen Modellwerten, expliziten Selbstangaben des Modells sowie mithilfe konsistenzbasierter Methoden verlässliche Indikatoren für die Qualität und Sicherheit während der Steuerung ableiten lassen. Zur Analyse werden realitätsnahe Instruktionen und Benutzereingaben in Form von Kundenanfragen verwendet. Für jede Anfrage liegen Referenzlösungen vor, anhand derer die Qualität der Modellantworten beurteilt werden kann.

Die Ergebnisse dieser Arbeit sollen Aufschluss darüber geben, welche bestehenden Verfahren am besten dazu geeignet sind, zu erkennen, wie sicher sich LLMs in ihren Antworten und vorgeschlagenen Aktionen sind und wie diese Erkenntnisse für den sicheren, praktischen Einsatz von Sprachmodellen in der Prozessautomation nutzbar gemacht werden können. Richtungsweisende Arbeiten im Bereich der Unsicherheitsquantifizierung sind Shorinwa et al., 2025 durch ihren umfangreichen Überblick bestehender Unsicherheitsuntersuchungen, Manakul et al., 2023, die in ihrer Veröffentlichung neben ihrem samplingbasierten Ansatz mit „SelfCheckGPT“ auch ähnliche Metriken beleuchten, sowie Lin et al., 2022, die in ihrer Arbeit den Ansatz der selbst verbalisierten Unsicherheitserhebung präsentieren.

1.1. Zielsetzung der Arbeit

Während der Themenfindung für diese Arbeit eröffnete sich die Möglichkeit diese Thesis in Kooperation mit der Firma „Absolute Software GmbH“ (Kapitel 1.3) zu schreiben. Das praktische Ziel war es, ein KI-System zu entwickeln, welches in der Lage sein soll, Anwendungen und Webseiten innerhalb eines Webbrowsers zu bedienen und zu automatisieren. Als konkrete Anwendung, die gesteuert werden soll, dient das Prozessmanagementtool „TSC-Online“ (Kapitel 3.1). Als während der Entwicklungsarbeit die ersten erfolgreichen Automatisierungsdurchläufe erzielt wurden, stellte sich zugleich die Frage, wie man bewerten kann, ob sich das System bei dem, was es tut, überhaupt sicher ist, denn überzeugend und selbstbewusst wirken, können große Sprachmodelle gut. Künstliche Intelligenz und große Sprachmodelle, wie OpenAIs ChatGPT oder Googles Gemini sind mittlerweile im Alltag angekommen und für die meisten Menschen ein Begriff. Viele Menschen vertrauen bei alltäglichen Fragen auf LLMs und sind dabei schon einmal auf halluzinierte oder fehlerhafte Antworten gestoßen. So berichtete selbst die Tagesschau, dass künstliche Intelligenz jede dritte Antwort erfindet: „(...) Doch dieses Vertrauen ist riskant, wie eine aktuelle Studie der Europäischen Rundfunkunion (EBU) zeigt. Der Zusammenschluss von 68 öffentlich-rechtlichen Sendern aus 56 Ländern hat systematisch die Zuverlässigkeit der populärsten KI-Systeme getestet. Das erschreckende Ergebnis: ChatGPT, Gemini und andere Chatbots erfinden bis zu 40 Prozent ihrer Antworten und stellen sie als Fakten dar.“ (Schieb, 2025).

Large Language Models stellen weit mehr als eine temporäre Trenderscheinung dar und sie einzusetzen bietet zahlreiche neue technische Möglichkeiten. Jedoch sollten die von ihnen generierten Antworten nicht als unstrittige Fakten gewertet, sondern Möglichkeiten etabliert werden, um halluzinierte und faktisch inkorrekte von wahren Antworten unterscheiden zu können. Letztlich sollten wir in der Lage sein, die Sicherheit eines Sprachmodells und seiner präsentierten Antwort evaluieren zu können und festzustellen, wenn sich ein Sprachmodell eigentlich gerade sehr unsicher ist.

Die Arbeit hat daher zum Ziel, die Unsicherheit von Large Language Models im Bereich der webbasierten Prozessautomation und Systemsteuerung zu untersuchen und dabei verschiedene Methoden anzuwenden, um die Unsicherheit von LLMs bei erzeugten Aktionen analysieren zu können. Dies geschieht durch interne Wahrscheinlichkeiten (Logprobs), über explizite, verbalisierte Selbstangaben zur eigenen Konfidenz, als auch konsistenzbasierte Ansätze, bei denen mehrere Antworten generiert und auf Konsistenz untersucht werden.

1.2. Aufbau der Arbeit

Um Lesern und Leserinnen einen Einstieg und allgemeinen Überblick über die inhaltlichen und wissenschaftlichen Themen dieser Arbeit zu bieten, wird der Rahmen und das Forschungsziel in dem [ersten Kapitel](#) näher ausgeführt. Kapitel [2](#) liefert ein umfassendes Verständnis über die theoretischen Grundlagen dieser Arbeit. Da es in dieser Thesis im Kern immer um moderne Sprachmodelle geht, werden zu Beginn die revolutionäre Transformer-Technologie von Vaswani et al., [2017](#) und wichtige Begriffe und Konzepte aus dem Feld der Large Language Models vorgestellt, woraufhin die Untersuchungsgrundlagen der Unsicherheitsquantifizierung detailliert beschrieben und relevante Arbeiten aus der Forschung präsentiert werden. Kapitel [3](#) liefert eine Übersicht über die Entwicklung und Implementierung der Anwendung. Kapitel [4](#) umfasst die Methodik und die verwendeten Metriken, um die Forschungsfrage dieser Arbeit letztlich zu beantworten und beschreibt außerdem, wie die Messgrößen erhoben werden, bevor deren Ergebnisse im folgenden Kapitel [5](#) übersichtlich dargestellt werden. Die Diskussion der Ergebnisse, mögliche Verzerrungen und Ausblick auf weiterführende Forschungsansätze finden daraufhin in Kapitel [6](#) statt. Die Arbeit schließt mit einem Fazit in Kapitel [7](#) ab.

1.3. Vorstellung des Unternehmens

Die Firma *Absolute Software GmbH*¹ ist ein seit 1995 bestehendes, inhabergeführtes IT-Unternehmen mit Sitz in Hamburg. Das Unternehmen bietet Beratungsleistungen und entwickelt Softwarelösungen im Bereich moderner Informationstechnologien an. Zu den Auftraggebern zählen neben mittelständischen Unternehmen auch internationale Konzerne. Mit mehr als 25 Jahren Erfahrung verfügt die Firma über ein breites technologisches Spektrum und unterstützt Organisationen bei digitalen Transformationen und individuellen IT-Anforderungen. Die Tätigkeitsschwerpunkte liegen in der Konzeption und Entwicklung maßgeschneiderter Software sowie in der strategischen Beratung zu IT-Infrastrukturen und Datensicherheit. In den vergangenen Jahren hat sich Absolute Software GmbH verstärkt auf die Themen Webentwicklung, Prozessautomatisierung, Datenanalyse, Virtual Reality, Augmented Reality, Digital Twins, 3D-Scanning und Künstliche Intelligenz ausgerichtet und war an über 500 internationalen Software- und Digitalisierungsprojekten beteiligt.

Die Arbeitsweise umfasst in der Regel eine ganzheitliche Betreuung, beginnend mit der Bedarfsanalyse über die Entwicklung und Implementierung der letztlichen Lösung hin zur weiterführenden Kundenbetreuung auch nach der Inbetriebnahme eines Produktes. Dabei findet ein kontinuierlicher Austausch mit den Kunden statt um die Lösungen an spezifische Geschäftsprozesse und Anforderungen anzupassen.

¹<https://absolute.de/>

2. Theoretische Grundlagen

2.1. Large Language Models

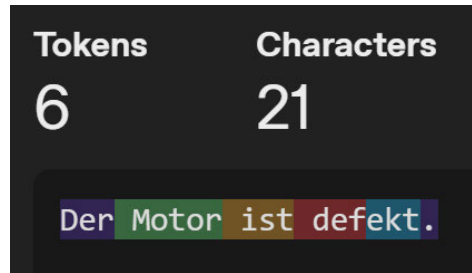
Das folgende Kapitel soll einen grundlegenden Überblick über die heute weit verbreiteten Large Language Models liefern. Neuronale Netze oder auch Künstliche Neuronale Netze (KNN) sind der Kern moderner Künstlicher Intelligenz (KI), des Maschinellen Lernens (ML) und damit letztlich auch von LLMs. Sie sind eine spezielle Art von Computerarchitektur, welche von der biologischen Funktionsweise des menschlichen Gehirns inspiriert sind. Wie auch das menschliche Gehirn mit Milliarden vernetzter Nervenzellen Daten verarbeitet, arbeiten KNNs sehr ähnlich und erkennen Muster und liefern eine Ausgabe. Auf Grundlage von Trainingsdaten und Erfahrungen können sie lernen und speziell an bestimmte Aufgaben, wie zum Beispiel die Klassifizierung von Bildern, das Generieren und Übersetzen von Texten, autonomes Fahren oder Gesichtserkennung angepasst werden.

Das Konzept Neuronaler Netze wurde erstmals in einer Arbeit im Jahr 1943 vorgestellt und in den 1950er und 1960er Jahren versuchten Informatiker einfache Neuronale Netze zu konstruieren. Danach fand es erst in den 1980er und 1990er Jahren wieder Anwendung in der Forschung der Künstlichen Intelligenz (IBM, [2021](#)). Der letzte Durchbruch, der dazu führte, dass künstliche Intelligenz im heutigen Alltag allgegenwärtig ist, geschah im Jahr 2017 durch die Entwicklung von Transformer-Modellen und dem Mechanismus der Selbstaufmerksamkeit (Self-Attention) durch die Arbeit von Vaswani et al., [2017](#).

2.1.1. Tokens und Embeddings

Bevor wir uns den Transformer-Modellen und deren Funktionsweise anschauen, sollten wir zuerst klären, wie Text und Sprache überhaupt verarbeitet werden. Der Begriff Sprachverarbeitung (Natural Language Processing, NLP) hat eine lange Historie und insbesondere im Kontext von Maschinellen Lernen und KI viele Anwendungsgebiete, darunter Übersetzungen, Zusammenfassungen, Textklassifizierung und heute vor allem Textgenerierung.

Text liegt immer in Form von sequentiellen Daten mit variabler Länge vor, welcher zuerst in eine numerische Vektorrepräsentation überführt werden muss, um von neuronalen Netzen



Abbildungung 2.1.: Resultierende Tokens für den Eingabesatz „Der Motor ist defekt.“ bei Verwendung des OpenAI Tokenizers für GPT-4o, Quelle: <https://platform.openai.com/tokenizer>

verarbeitet werden zu können (J. Yang et al., 2024). Ein wesentlicher Schritt darin liegt in der Tokenisierung (Tokenization), wobei die Bestandteile eines Textes in einzelne Tokens zerlegt werden. Vereinfacht kann man sich vorstellen, dass jedes Wort eines Satzes einem Token entspricht. Jedoch sind in der Realität auch feingranulare Subwörter oder Zeichen möglich. In Abbildung 2.1 wird gezeigt, wie der GPT-4o Tokenizer von OpenAI den Satz „Der Motor ist defekt.“ in einzelne Token zerlegt. Es ist zu sehen, dass aus dem Wort „defekt“ zwei Tokens resultieren, nämlich „def“ und „ekt“. Das ist eine wesentliche Eigenschaft von sogenanntem Byte Pair Encoding (BPE), was heutzutage häufig Verwendung findet. Das Vokabular wird dabei auf Grundlage eines Trainingscorpus erlernt. Zu Beginn werden Worte in einzelne Buchstaben zerlegt, wodurch jeder Buchstabe einem einzelnen Token entspricht. Daraufhin werden aus vorkommenden Buchstabenkombinationen eigenständige Tokens erlernt, aus deren Kombination verschiedene Worte zusammengesetzt werden können (Kipp, 2025, Kap. 14).

Ein Tokenizer verfügt über ein endliches Vokabular, in welchem jedes Token über einen eindeutigen Index beziehungsweise eine Token-Id verfügt, welche man sich als Position in der Liste aller möglicher Tokens vorstellen kann. Die Vokabulargröße von Modellen liegt in der Regel zwischen 10.000 und 100.000.000 Einträgen, der in dieser Arbeit verwendete Tokenizer für GPT-4o hat laut J. Yang et al., 2024 ein Vokabular von 199.997 Tokens. Der in Abbildung 2.1 gezeigte Satz kann letztlich durch folgende Token-IDs repräsentiert werden: [13518, 21702, 2496, 1056, 7323, 13] (Kipp, 2025, Kap. 14).

Während Tokens diskrete Symbole sind, benötigen neuronale Modelle kontinuierliche numerische Eingaben. Genau hier setzen Embeddings an. Ein Embedding ordnet jedem Token einen Vektor aus reellen Zahlen zu und repräsentiert ihn in einem mehrdimensionalen Raum. Jeder Vektor eines Tokens verfügt über m Elemente (sogenannte Features), welche beliebige Dezi-malwerte annehmen können. m wird auch als die Embedding-Dimensionen bezeichnet. Diese Vektorrepräsentationen, welche jedes Token in einen Vektorraum projizieren, werden durch Training erlernt. Die einzelnen Zahlen eines Embedding-Vektors haben für sich genommen

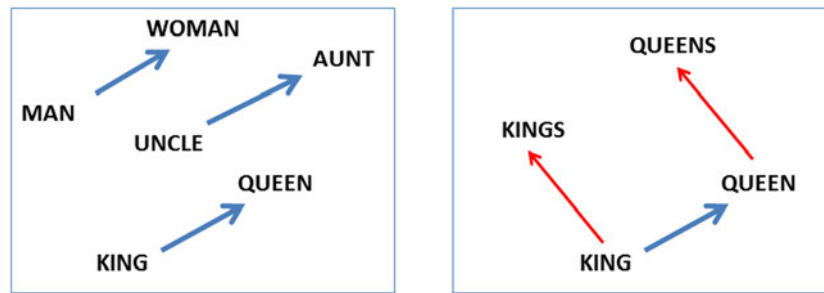


Abbildung 2.2.: Links: Zeigt den Vektoroffset von drei Wortpaaren anhand des Geschlechts-Features, Rechts: Zeigt die Projektionen zweier Wortpaare anhand ihrer Singular/Plural Relation, Quelle: Mikolov et al., 2013

keine intuitive Bedeutung. Ihre Aussagekraft entsteht erst im Zusammenspiel mit anderen Embeddings. So sind beispielsweise die Embeddings von „Hund“ und „Katze“ typischerweise ähnlicher zueinander als die von „Hund“ und „Tisch“. Diese Eigenschaft erlaubt es Modellen, Bedeutungszusammenhänge zwischen verschiedenen Tokens zu erfassen. Jedes Feature kann als semantischer Aspekt verstanden werden (Kipp, 2025, Kap. 14.4), sodass die Kategorie „Tier“ vereinfacht ein Feature mit hohem Wert für die Worte „Hund“ und „Katze“ sein könnte. In der Realität weiß man jedoch nie, wofür die einzelnen Features genau stehen, da sie während des Trainings automatisch erlernt werden. Besteht ein Wort aus mehreren Tokens, so ergibt sich das letztliche Wortembedding aus dem Durchschnitt der einzelnen Tokenembeddings. Ähnlich können sogar Embeddings für ganze Sätze oder Dokumente erzeugt werden.

In ihrer Arbeit stellten Mikolov et al., 2013 heraus, dass Vektorrepräsentationen auch syntaktische und semantische Charakteristiken überraschend gut widerspiegeln. So ist in Abbildung 2.2 zu sehen, dass das Geschlecht in dem Vektoroffset zwischen den Worten „Man“ und „Woman“ abgebildet ist. Addiert man den Vektoroffset beispielsweise zu dem Embedding des Wortes „Uncle“, so landet man nah bei dem Embedding des Wortes „Aunt“.

Text und Sprache werden also so verarbeitet, dass sie zunächst in Tokens zerlegt und anschließend als Embeddings in numerische Vektorrepräsentationen überführt werden, sodass neuronale Netze ihre Bedeutungen und Zusammenhänge erfassen können.

2.1.2. Transformer-Modelle

Häufig werden im allgemeinen Sprachgebrauch die Begriffe Transformer und Large Language Model austauschbar und gleichbedeutend behandelt. Technisch gesehen stimmt das nicht immer, denn ein LLM muss nicht zwangsläufig auf der Transformer-Technologie bestehen und ein Transformer muss nicht zwangsläufig ein LLM sein. Im Kontext dieser Arbeit ist im Allgemeinen jedoch immer von transformerbasierten LLMs die Rede.

Transformer-Modelle wurden erstmals in der Arbeit „Attention Is All You Need“ von Vaswani et al., 2017 vorgestellt. Sie basieren auf Aufmerksamkeitsmechanismen (Attention) und ersetzen damit zentrale Mechanismen früherer Architekturen¹. Ihre Ausgaben liefern eine höhere Qualität in Bezug auf sprachliche Abhängigkeiten. Zugleich können Eingabesequenzen parallel verarbeitet werden, wodurch sie eine sehr hohe Effizienz aufweisen.

Das ursprüngliche Transformer-Modell bestand, wie der Großteil gängiger neuronaler Netze zu Übersetzungszwecken, aus einer Encoder-Decoder-Struktur. Bleibt man im Bereich der Übersetzungen von einer Sprache in eine andere, so hat der Encoder die Aufgabe den zu übersetzenden Satz zu tokenisieren und anschließend in Embeddings umzuwandeln. Der Decoder erhält diese Embeddings als Eingang und erstellt daraus letztlich eine Sequenz von Ausgangstokens, in diesem Fall die Übersetzung des Satzes in die Zielsprache. Jedes Token wird der Reihe nach generiert und daraufhin der zu verarbeitenden Eingangssequenz angehängt, man spricht daher von einem autoregressiven Modell (Vaswani et al., 2017). Heute existieren neben Encoder-Decoder Modellen auch Encoder-only und Decoder-only Modelle. Letztere sind insbesondere bei den OpenAI Modellen weit verbreitet (Minaee et al., 2025). Im Folgenden werden die zentralen Elemente und Funktionen von Transformer-Modellen näher beschrieben, da sie einerseits die zentrale technologische Grundlage dieser Arbeit bilden und andererseits das Verständnis dafür schafft, wie die im weiteren Verlauf betrachteten Tokenwahrscheinlichkeiten entstehen.

Self-Attention

Wie oben angedeutet, bildet Attention das Herzstück der Transformer-Technologie. Attention beantwortet im Kern die Frage, auf welche anderen Wörter oder Tokens innerhalb einer Sequenz geachtet werden soll, um das aktuelle Wort korrekt zu verstehen. Anhand des folgenden Beispiels wird dies etwas klarer: „The animal didn’t cross the street because it was too tired“. Einem Menschen wird aus dem Kontext klar, dass sich das Wort „it“ auf „the animal“ bezieht und nicht auf „the street“, jedoch nicht einer Maschine. Attention liefert Transformer-Modellen also einen Weg ein Verständnis von Worten und Tokens zueinander zu erlangen (Alammar, 2018). Das Konzept der Attention wurde erstmals von Bahdanau et al., 2016 vorgestellt und galt als großer Fortschritt im Bereich der Sprachübersetzung mithilfe neuronaler Netze.

Um die Self-Attention zu berechnen, werden zu Beginn für jedes Token einer Eingangssequenz drei unterschiedliche Projektionen erzeugt: Query(q), Key(k) und Value(v). Diese Repräsentationen stammen aus derselben Eingangssequenz, werden aber durch unterschiedliche, trainierte Gewichtungsmatrizen (W_q , W_k , W_v) erzeugt. Sie können vereinfacht als drei

¹Auf eine detaillierte Betrachtung von KNNs wird an dieser Stelle verzichtet. Für eine vertiefte Auseinandersetzung empfehle ich die Arbeit von Schlüter, 2016.

verschiedene Rollen desselben Tokens verstanden werden. Der Query-Vektor eines Tokens beschreibt, welche anderen Tokens für seine Kontextrepräsentation relevant sind. Die Key-Vektoren anderer Tokens geben an, welche Informationen sie anbieten. Die Value-Vektoren enthalten wiederum Information, die letztlich gewichtet in die Ausgabe einfließen.

Dies soll am Beispiel der Self-Attention für das Token „journey“ im Satz „Your journey starts with one step“ (Abbildung 2.3) veranschaulicht werden. Für jedes Token wird ein Attention-Score ω berechnet. Dieser gibt an, wie stark es auf die anderen Tokens der Sequenz achten soll, indem man den Query-Vektor eines Tokens mit den Key-Vektoren der anderen Tokens vergleicht. Um aus diese Attention-Scores Gewichte zu machen, müssen sie normalisiert werden. Der wichtigste Schritt dabei ist die Anwendung der Softmax-Funktion:

$$\text{softmax}(\omega)_i = \frac{e^{\omega_i}}{\sum_{j=1}^n e^{\omega_j}} \quad (2.1)$$

Ihr Vorteil ist, dass sie eine Liste mit beliebig positiven oder negativen Werten auf Werte zwischen 0 und 1 skaliert und die Summe aller Ergebniswerte genau 1 entspricht. Die Ergebnisse können als Wahrscheinlichkeiten verstanden werden. Die daraus resultierenden Attention-Weights α geben an, wie wichtig welche anderen Tokens für das aktuelle Token sind (Vaswani et al., 2017, Alammari, 2018). Für jedes Token werden diese Gewichte dann mit ihren Value-Vektoren multipliziert, wodurch die Relevanz jedes Tokens für das aktuelle Token bewertet wird. Die Summe dieser Ergebnisse nennt man Kontextvektor, welcher letztlich das Ergebnis der Self-Attention eines Tokens ist und im Anschluss in das Feed-Forward Netzwerk weitergegeben wird (Raschka, 2025, S. 65-70).

In Wirklichkeit müssen diese Berechnungen jedoch nicht nur für eines, sondern für alle Token der Eingangssequenz durchgeführt werden. Aufgrund einer einfacheren und schnelleren Verarbeitung geschieht diese Berechnung durch Matrizen. Außerdem führt das ursprüngliche Transformer-Modell diese Berechnung der Self-Attention nicht nur ein einziges Mal durch, sondern parallel auf acht sogenannten „Attention-Heads“ mit unterschiedlichen Gewichten. Diese Berechnung wird als „Multi-Head Attention“ bezeichnet (Vaswani et al., 2017). Durch die unterschiedlichen Gewichte erlangt das Modell die Fähigkeit, sich auf verschiedene Positionen eines Satzes zu fokussieren und Informationen aus unterschiedlichen Repräsentationsräumen oder „Blickwinkeln“ zu erfassen (Vaswani et al., 2017).

Feed-Forward Netzwerk

Der generelle Aufbau eines Transformer-Modells ist in Abbildung 2.4 dargestellt. Dort ist zu sehen, dass jedem Attention-Block ein Feed-Forward Netzwerk (FFN) folgt. FFNs sind neuronale Netze, bei denen Informationen immer nur vorwärts transportiert werden. Dieses FFN be-

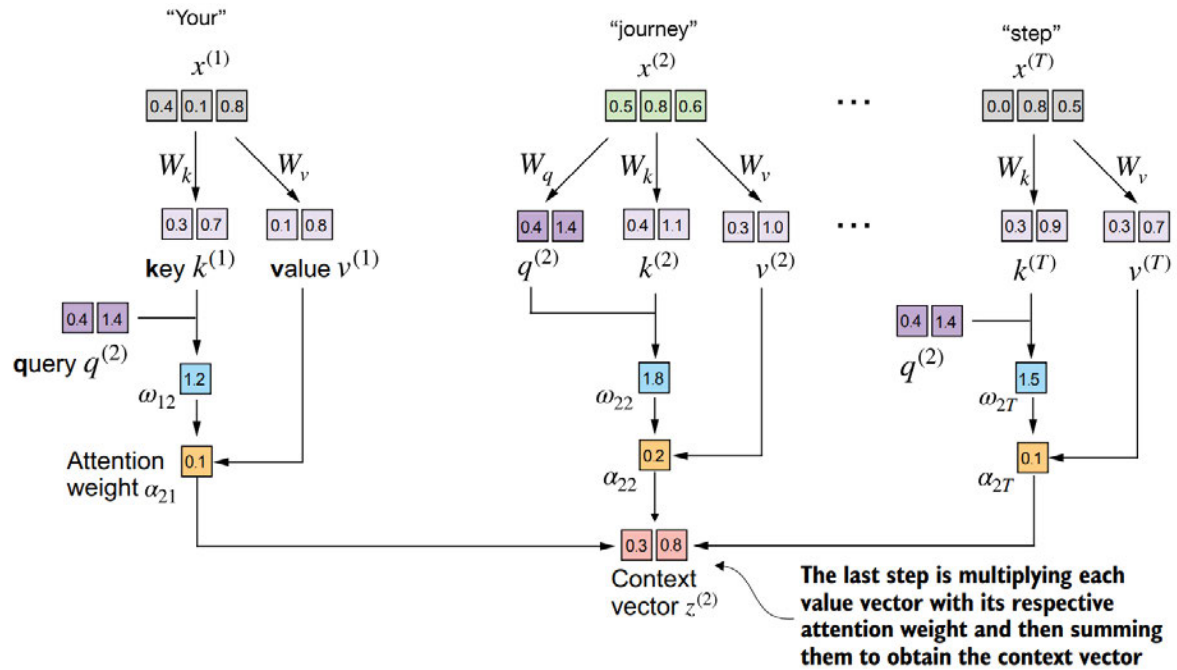


Abbildung 2.3.: Die Berechnung der Self-Attention und damit des finalen Kontextvektors z am Beispiel des zweiten Tokens „journey“, Quelle: Raschka, 2025, S. 69

steht insbesondere aus zwei linearen Transformationen mit einer ReLU-Aktivierungsfunktion dazwischen (Vaswani et al., 2017). ReLU (Rectified Linear Unit) ist eine lineare Funktion, die den Eingangswert ausgibt, wenn er positiv ist und 0 ausgibt, wenn der Eingang negativ ist. Das FFN spielt eine wesentliche Rolle dabei, die Fähigkeit des Modells zu verbessern aus Daten zu lernen und daraus zu generalisieren, es extrahiert Informationen und Wissen (Raschka, 2025, S. 100-109).

Decoder

Bis jetzt haben wir uns mit den Encodern beschäftigt (siehe Abbildung 2.4 links), die die Eingangssequenz verarbeiten und an die Decoder (rechts) weitergeben. Die oben beschriebenen Elemente finden sich auch im Decoder wieder. Die Ausgabe des letzten Encoders wird in Aufmerksamkeitsvektoren K und V transformiert und jedem Decoder-Block als Eingang zugeführt. Sie unterstützen die Decoder dabei, auf welche Positionen sie sich konzentrieren sollen. Genau wie die Encoder, gibt auch jeder Decoder sein Ergebnis an den folgenden Decoder weiter. Der letzte Decoder eines Zeitschritts gibt sein Ergebnis wiederum dem untersten Decoder im nächsten Zeitschritt als Eingang weiter (Alammar, 2018).

Zuvor wurde bereits beschrieben, dass Self-Attention für jedes Token Attention-Gewichte zu allen Tokens der Sequenz berechnet. Anders als im Encoder, erzeugt der Decoder Tokens

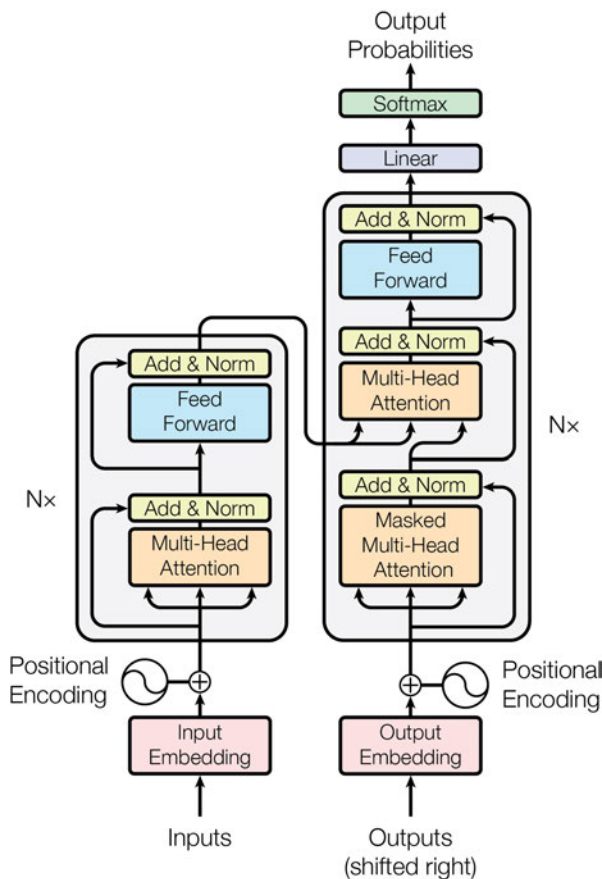


Abbildung 2.4.: Die Architektur des Transformer-Modells wie es erstmals von Vaswani et al. vorgestellt wurde. Auf der linken Seite befindet sich der Encoder, der die Eingabe verarbeitet und auf der rechten Seite der Decoder, welcher die Ausgabe generiert. Quelle: Vaswani et al., 2017

autoregressiv. Damit der Transformer lernen kann, dürfen zukünftige Tokens späterer Zeitschritte der Zielsequenz nicht berücksichtigt werden, da dies wie ein Blick in die Zukunft wäre. Deshalb werden Tokens späterer Zeitschritte während der Self-Attention Berechnung maskiert und daher nicht berücksichtigt. Diese sogenannte „Masked Self-Attention“ im Decoder verhindert letztlich, dass ein Token auf zukünftige Tokens zugreifen kann, damit der Transformer seine autoregressive Eigenschaft beibehält (Vaswani et al., 2017).

Ausgabeschicht

Wurden alle Decoder durchlaufen und die Ausgabe des letzten Transformerblocks liegt vor, so wird diese der linearen Ausgabeschicht eingespeist. Diese transformiert die Decoder-Ausgabe in einen höher-dimensionalen Raum, dessen Dimensionen der Größe des verwendeten Vokabulars entspricht. Das Vokabular von GPT-2 entsprach beispielsweise einer Größe von 50257 Einträgen, dementsprechend projizierte die lineare Schicht den Eingang auch in einen Vektor mit 50257 Elementen, wobei jeder Eintrag einem Token des Vokabulars zugeordnet ist (Raschka, 2025, S. 117). Diesen Vektor nennt man auch den „Logits-Vektor“. Zuletzt wird auf diesen Logits-Vektor die Softmax-Funktion (Formel 2.1) angewendet, um die rohen

Gleitkommazahlen in Wahrscheinlichkeiten umzuwandeln. Somit erhält jedes Token des Vokabulars eine Wahrscheinlichkeitsvorhersage, um als nächste Ausgabe gewählt zu werden (Alammar, 2018). Abschließend sei angemerkt, dass moderne LLMs die Ausgaben häufig nicht direkt als Wahrscheinlichkeiten, sondern als Logarithmen der Wahrscheinlichkeiten (Logprobs) ausgeben, da diese numerisch stabiler sind und sich mathematisch effizienter weiterverarbeiten lassen (Raschka, 2025, S. 137-138).

2.1.3. Textgenerierung

In Kapitel 2.1.2 wurde beschrieben, wie ein Transformer-Modell, basierend auf einer Eingangssequenz, Schritt für Schritt Ausgabetokens erzeugt. Speist man beispielsweise die Eingangssequenz „Hello, I am“ in ein LLM, wie in Abbildung 2.5 dargestellt, so wird im ersten Durchgang eine Wahrscheinlichkeitsverteilung für das nächste Token berechnet und ausgewählt, in diesem Fall ist es das Token „a“. Genau genommen beinhaltet die Vorhersage nur die ID des Tokens, welche in einem zusätzlichen Schritt in das zugehörige Token übersetzt werden muss. Dieses Ausgabetoken wird der Eingangssequenz angehängt, welche nun im nächsten Durchgang als Eingabe dient. In vielen Durchgängen dieses Prozesses werden immer weiter Tokens erzeugt, bis eine Ausgabesequenz vollständig erzeugt oder eine vorgegebene maximale Anzahl von Ausgabetokens erzeugt wurde. In diesem Fall lautet sie „Hello, I am a model ready to help.“ (Raschka, 2025, S. 122-127).

Decoding Strategien

Es gibt verschiedene Techniken, wie entschieden wird, welches Token letztendlich ausgewählt wird. Der einfachste Fall nennt sich „Greedy Decoding“. Dabei wird einfach das Token mit der höchsten Wahrscheinlichkeit aus der Wahrscheinlichkeitsverteilung gewählt. Mit Greedy Decoding würde die Ausgabesequenz bei dem Beispiel in Abbildung 2.5 für die Eingabe „Hello, I am“ immer gleich aussehen. Der Ausgabetext ist dadurch wenig vielseitig.

Anstatt immer das Token mit der höchsten Wahrscheinlichkeit auszuwählen, kann durch „Temperature Scaling“ die Zufälligkeit und Variabilität der Ausgabe gesteuert werden. Dazu werden die rohen Tokenlogits vor der Softmax Funktion (Formel 2.1) in der Ausgabeschicht durch einen Skalierungsfaktor T geteilt, was mathematisch durch

$$\text{softmax}(\omega)_i = \frac{e^{\omega_i/T}}{\sum_{j=1}^n e^{\omega_j/T}} \quad (2.2)$$

ausgedrückt werden kann. Eine Temperatur von $T = 1$ lässt die ursprünglichen Logits und damit die daraus resultierende Wahrscheinlichkeitsverteilung unverändert. Wird eine Temperatur von $T < 1$ gewählt, führt dies zu einer spitzeren Verteilung. Das wahrscheinlichste

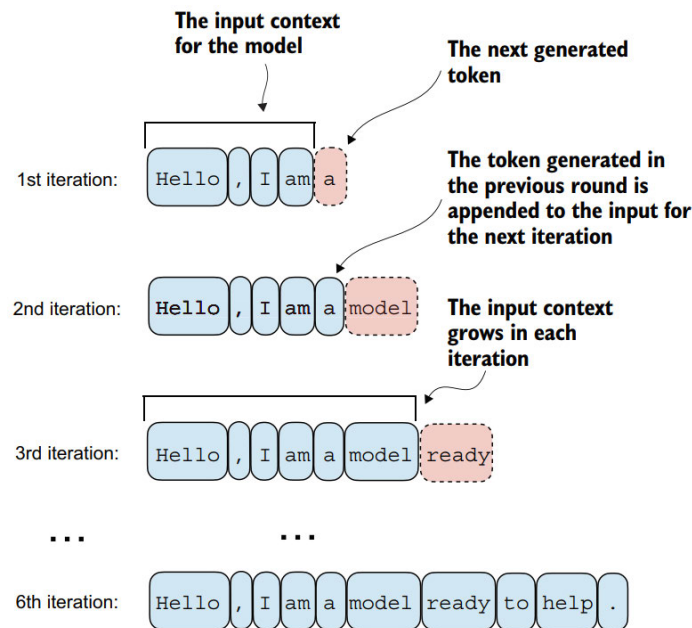


Abbildung 2.5.: Beispiel der autoregressiven Texterzeugung. Beginnend mit der Eingabesequenz „Hello, I am“ wird im ersten Schritt das Token „a“ generiert und der Sequenz angehängen. Die Eingabesequenz im zweiten Durchgang lautet demnach „Hello, I am a“. Diese Schritte werden wiederholt bis die Ausgabe­sequenz vollständig generiert wurde. Quelle Raschka, 2025, S. 122

Token bekommt eine noch höhere Priorität, während sich die Wahrscheinlichkeiten unwahrscheinlicher Tokens null nähern. Eine Temperatur von $T > 1$ führt wiederum zu einer flacheren, gleichmäßigeren Verteilung der Wahrscheinlichkeiten über die Tokens, wodurch die Wahrscheinlichkeiten eigentlich unwahrscheinlicher Tokens wächst, was zu kreativen aber auch unsinnigen Ergebnissen führen kann (Raschka, 2025, S. 151-157).

Eine weitere Sampling-Strategie, die das Auftreten von unsinnigen Ergebnissen reduziert, nennt sich „Top-k Sampling“. Dabei wird die Auswahl auf die k Tokens mit den höchsten Wahrscheinlichkeiten begrenzt, während alle anderen Tokens von der Auswahl ausgeschlossen werden. In der Praxis wird dies umgesetzt, indem alle Logits außer den oberen k Tokens vor der Softmax-Funktion auf $-\infty$ gesetzt werden. Durch die Softmax-Funktion ergibt sich dann für diese Tokens eine Wahrscheinlichkeit von 0, während die Summe der übrigen k Tokens 1 ergibt (Raschka, 2025, S. 155-157). Neben Top-k Sampling, stellt „Top-p Sampling“ eine weitere Alternative dar, bei welcher stattdessen nur aus den höchsten Tokens ausgewählt wird, deren Summe aus Wahrscheinlichkeiten einen Grenzwert p überschreitet (Raschka, 2025, S. 295).

Die letzte Sampling-Strategie, die hier vorgestellt werden soll, nennt sich „Beam Search“. Sie ist ein weit verbreitetes Verfahren, bei denen nicht eine einzige Sequenz, sondern eine

Menge von B Sequenzen („Hypothesen“) erzeugt werden. Statt einer Sequenz werden also die wahrscheinlichsten B Sequenzen erzeugt. Zu jedem Zeitschritt wird jede Sequenz um alle möglichen Tokens erweitert, von denen die top- B Tokens ausgewählt werden. Dieses Vorgehen wird iterativ bis zum Ende der Sequenz wiederholt, woraufhin die wahrscheinlichste Sequenz anhand seiner Log-Wahrscheinlichkeiten als Ausgabe gewählt wird (Vijayakumar et al., 2018).

2.2. Prompt Engineering

Im Gegensatz zu traditionellen, ressourcenintensiven Techniken, wie dem Finetuning, wobei ein bereits vortrainiertes Modell auf einen speziellen Datensatz oder eine bestimmte Aufgabe weitertrainiert wird, liefert „Prompt Engineering“ eine leichtgewichtige Möglichkeit, zuvor trainierte Modelle zu einem gewünschten Verhalten zu instruieren. Prompts bezeichnen strategische und aufgabenspezifische Anweisungen um das Modellverhalten gezielt zu steuern. Im einfachsten Fall wird Prompt Engineering in Form von natürlicher Sprache eingesetzt, jedoch bietet es auch die Möglichkeit das Wissen des Sprachmodells durch Vektorrepräsentationen zu erweitern (Sahoo et al., 2025). Prompts können von einfachen Fragen bis hin zu spezifischen Aufgaben reichen und komplexe Strukturen beinhalten. Im Kern liegt die höchste Anforderung an einen Prompt darin, ihn so zu gestalten, dass ein Sprachmodell eine ganz gezielte Ausgabe liefert (Minaee et al., 2025). Da die Landschaft des Prompt Engineerings mittlerweile sehr viele verschiedene Techniken umfasst, sollen die gängigsten Verfahren an dieser Stelle einmal vorgestellt werden.

Zero-Shot Prompting

Zero-Shot Prompting stellt die einfachste Technik des Promptings dar. Der Prompt umfasst lediglich eine Aufgabenbeschreibung, Frage oder Instruktion. Das Sprachmodell nutzt lediglich sein vorhandenes Wissen und generiert auf Grundlage des Prompts eine Ausgabe. Zu Beginn war diese Möglichkeit sehr besonders, da keine gelabelten Daten notwendig waren (Radford et al., 2019, Sahoo et al., 2025).

Few-Shot Prompting

Anders als beim Zero-Shot Prompting werden dem Sprachmodell bei dem „Few-Shot Prompting“ einige wenige Beispiele, die die gewünschte Ausgabe zu einem Eingabebeispiel demonstrieren, mitgeliefert (Brown et al., 2020). Mithilfe von ein paar ausgewählten Beispielen wurde die Modellleistung gegenüber keinen Beispielen nachweislich verbessert. Few-Shot

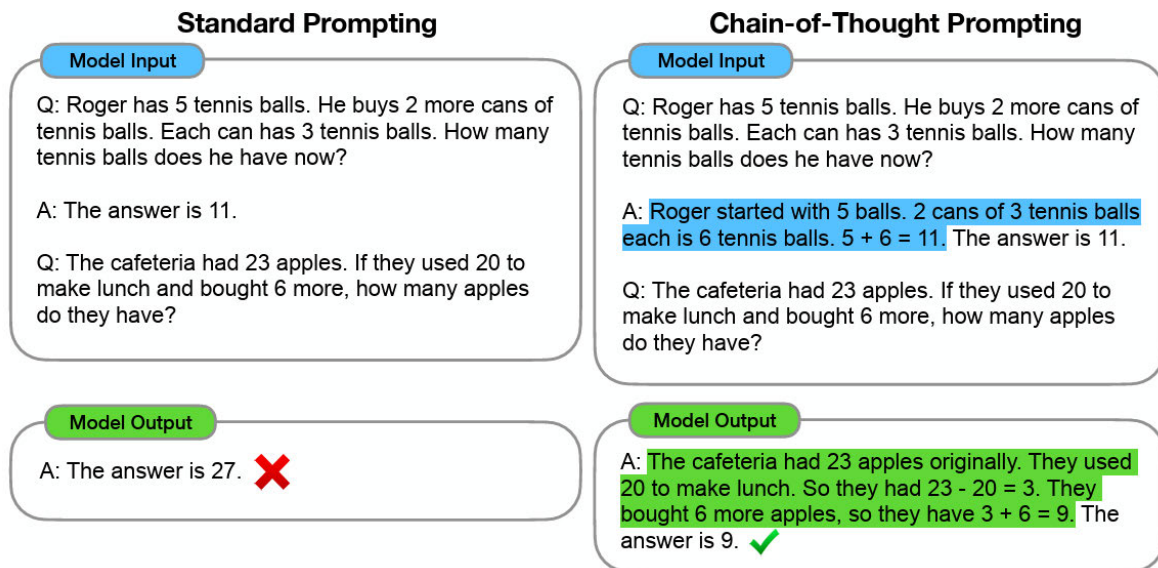


Abbildung 2.6.: Demonstration von Chain-of-Thought Prompting bei arithmetischen Aufgaben gegenüber Standard Prompting. Reasoning Prozesse wurden farblich hervorgehoben. Quelle: Wei et al., 2023

Prompting sollte jedoch mit Vorsicht angewandt werden, da die Auswahl von Beispielen die Antwort von Sprachmodellen beeinflussen kann. Beispielsweise können bestimmte Worte innerhalb der Beispiele auch in den generierten Antworten bevorzugt gewählt werden, wodurch ein unerwünschter Bias entstehen kann (Sahoo et al., 2025).

Chain-of-Thought Prompting

„Chain-of-Thought“ (CoT) ist von Wei et al., 2023 vorgestellte Technik, um die Hürden von Sprachmodellen insbesondere bei komplexen Aufgaben zu überwinden, welche mehrere Gedankenschritte erfordern. Dabei werden Sprachmodelle aufgefordert, bei dem Generieren einer Antwort Schritt für Schritt logisch zu denken. Logisches Denken wird im Kontext von LLMs auch „Reasoning“ genannt und ist daran angelehnt, wie auch Menschen Aufgaben in mehreren Schritten unterteilen bevor sie eine finale Antwort geben (Wei et al., 2023). Abbildung 2.6 zeigt wie ein Sprachmodell durch CoT eine korrekte Antwort auf eine arithmetische Frage schlussfolgern kann, auf die es ohne eine fehlerhafte Antwort gegeben hätte, indem es in einzelnen Schritten Teilergebnisse aufführt. Als weitere Vorteile führen Wei et al. auf, dass es durch CoT möglich ist, nachzuvollziehen und zu interpretieren, wie ein Sprachmodell zu seiner letztlichen Antwort gekommen ist, wodurch in gewisser Weise eine Art des Debuggings ermöglicht wird. Zudem kann CoT ohne technischen Aufwand, lediglich durch geeignete Aufforderung im Prompt, implementiert werden.

Self-Consistency

Auf dem Konzept des Chain-of-Thought Promptings bauen Wang et al., 2023 in ihrer Arbeit „Self-Consistency Improves Chain of Thought Reasoning in Language Models“ auf. Unter der Idee, dass jeder Mensch anders denkt und es meist verschiedene valide Wege und Ansätze gibt, ein Problem zu lösen, stellen sie die Methode der „Self-Consistency“ vor. Anstatt sich auf einen einzigen Reasoningpfad zu verlassen, der durch Greedy Decoding nur den Pfad mit den höchsten Tokenwahrscheinlichkeiten wählt, werden bei Self-Consistency mehrere Gedankenpfade erzeugt. Letztlich wird die finale Antwort ausgewählt, die am häufigsten von den verschiedenen Reasoningpfaden erzielt wurde und daher am konsistentesten gilt (Wang et al., 2023).

2.3. Unsicherheitsquantifizierung von LLMs

Große Sprachmodelle neigen nach wie vor zu falschen oder halluzinierten Aussagen, die sie mit hoher Überzeugung vortragen (Y. Yang et al., 2025, Y. Huang et al., 2025), was das Vertrauen in die generierten Antworten mildert. Im Bereich großer Sprachmodelle bildet die Unsicherheitsquantifizierung daher ein sehr umfangreiches und bislang in wesentlichen Teilen unbeantwortetes Forschungsfeld. Ein wesentliches Thema dieser Arbeit ist daher die Erkennung von Unsicherheit in den von LLMs generierten Antworten. Was Unsicherheit im Kontext von Large Language Models überhaupt ist und wie sie unterschieden wird, soll im Folgenden näher erläutert werden. Es sei angemerkt, dass sich diese Arbeit mit der Unsicherheitsquantifizierung in Bezug auf vortrainierte Modelle bezieht, insbesondere auf „Foundation Models“ wie OpenAIs ChatGPT. Ansätze, die die Unsicherheit zu Trainingszeiten untersuchen, werden daher nicht behandelt. Diverse Übersichtsartikel adressieren das Problem mangelnder Zuverlässigkeit von LLMs und vergleichen Methoden der Unsicherheitsquantifizierung, um die Vertrauenswürdigkeit von Modellantworten messbar zu machen (Shorinwa et al., 2025, H.-Y. Huang et al., 2024, Geng et al., 2024). Die zentralen Ansätze und Konzepte zur Messung und Evaluierung von Unsicherheit werden im Folgenden ausgeleuchtet.

An dieser Stelle ist anzumerken, dass die Forschung zu großen Sprachmodellen und ihren breiten Anwendungsgebieten gegenwärtig zu den dynamischsten und publikationsstärksten Forschungsfeldern überhaupt zählt. Aufgrund dieser rasanten Entwicklungen wurden Veröffentlichungen ab August 2025 nicht mehr berücksichtigt.

2.3.1. Unsicherheit und Konfidenz

Im Kontext von LLM-generierten Antworten bezeichnet Unsicherheit (uncertainty) das Ausmaß des Nichtwissens oder Zweifels eines Modells bezüglich der Richtigkeit seiner Antwort. Konfidenz (confidence) hingegen meint die vom Modell angenommene Sicherheit, seine Antwort sei korrekt. Häufig interpretiert man Konfidenz als die Wahrscheinlichkeit, mit der die angegebene Antwort korrekt ist, besonders in der Kombination von Prompt und Antwort. Unsicherheit wird dabei häufig als Gegenstück angesehen, wobei eine hohe Unsicherheit für ein niedriges Vertrauen und niedrige Unsicherheit für ein hohes Vertrauen in die Antwort verstanden wird. In vielen Arbeiten werden beide Begriffe daher als zwei Seiten derselben Medaille oder sogar synonym betrachtet (Geng et al., 2024, Shorinwa et al., 2025). Wichtig ist jedoch zu erwähnen, dass eine hohe Konfidenz nicht garantiert, dass eine Antwort faktisch korrekt ist, sondern nur die vom Modell empfundene Sicherheit widerspiegelt. Große Sprachmodelle präsentieren ihre Antworten meist mit hoher Überzeugung, wobei es durchaus vorkommt, dass Antworten, die mit großer Sicherheit vorgetragen werden, faktisch inkorrekt sind.

In der Literatur wird in der Regel zwischen zwei Arten von Unsicherheit unterschieden: aleatorische und epistemische Unsicherheit.

Aleatorische Unsicherheit bezeichnet die Ungewissheit, die durch inhärente Zufälligkeit im zugrundeliegenden Prozess entsteht. Sie wird als nicht reduzierbare Unsicherheit verstanden, da sie selbst mit umfangreicherem Wissen nicht vollständig eliminiert werden kann. Klassische Beispiele sind der Würfelwurf oder Münzwurf, wobei der Ausgang vom Zufall abhängt. Übertragen auf LLMs bedeutet aleatorische Unsicherheit, dass Fragen auch mit mehr Trainingsdaten oder Größe des Sprachmodells selbst nicht mit höherer Gewissheit beantwortet werden können (Shorinwa et al., 2025). Modellierer haben keine Möglichkeit aleatorische Unsicherheit mit weiteren Daten zu reduzieren (Kiureghian und Ditlevsen, 2009).

Epistemische Unsicherheit hingegen resultiert aus mangelndem Wissen des Modells. Anders als aleatorische Unsicherheit ist diese durch beispielsweise mehr Trainingsdaten oder Wissen prinzipiell reduzierbar (Kiureghian und Ditlevsen, 2009). Epistemische Unsicherheit liegt dann vor, wenn ein LLM mit einer Frage konfrontiert wird, zu der in den Trainingsdaten wenig oder keine Informationen vorhanden waren. Das Modell hat also Unwissen oder Wissenslücken, um eine sichere Antwort zu geben (Shorinwa et al., 2025).

2.3.2. Tokenwahrscheinlichkeitsbasierte Unsicherheitsquantifizierung

Bei der Untersuchung der Unsicherheit von LLMs kann man generell zwischen Whitebox- und Blackbox-Ansätzen unterscheiden. Man spricht von Whitebox Systemen, wenn die zugrundeliegende Architektur eines Modells öffentlich zugänglich ist, sodass interne Zustände, wie zum Beispiel die Wahrscheinlichkeitsverteilungen der generierten Tokens einsehbar und nachvollziehbar sind (Shorinwa et al., 2025). Das Prinzip der Tokenerzeugung auf Grundlage der Wahrscheinlichkeitsverteilung wurde bereits in Kapitel 2.1.2 behandelt. Auf der Grundlage dieser verfügbaren Tokenwahrscheinlichkeiten jeder Position einer Ausgabe können verschiedene Metriken angewandt werden um letztlich eine Aussage über die Unsicherheit der generierten Antwort treffen zu können. Die folgenden Metriken stellen Whitebox-Ansätze dar, um die Unsicherheit auf dieser Grundlage zu evaluieren.

Es wird angemerkt, dass die logarithmischen Wahrscheinlichkeiten in der Literatur häufig als „Logprobs“ (logarithmic probabilities) bezeichnet werden. Wahrscheinlichkeiten werden dementsprechend zu Logprobs, indem man sie logarithmiert. Mithilfe der Exponentialfunktion werden Logprobs wieder zu Wahrscheinlichkeiten überführt, wie in der folgenden Funktion zu sehen. Dabei bezeichnet p die Wahrscheinlichkeit und L die logarithmische Wahrscheinlichkeit.

$$L = \log p \quad p = e^L \quad (2.3)$$

Durchschnittliche Token Log-Wahrscheinlichkeit

Diese Metrik betrachtet die Unsicherheit auf Grundlage der durchschnittlichen logarithmischen Token-Wahrscheinlichkeit aller Token eines Satzes. Dabei ist I die Anzahl aller Tokens im Satz und p_i die Wahrscheinlichkeit des Tokens an der Position i .

$$\text{Avg}(-\log p) = -\frac{1}{I} \sum_i \log p_i \quad (2.4)$$

Laut Manakul et al., 2023 stammen Unsicherheiten und Halluzinationen von Positionen mit niedriger Wahrscheinlichkeit und flacher Wahrscheinlichkeitsverteilung. Eine hohe durchschnittliche Wahrscheinlichkeit (bzw. ein niedriger Wert von $-\log p$) deutet auf hohe Konfidenz und niedrige Unsicherheit hin (Manakul et al., 2023, Y. Huang et al., 2025).

Maximale Token Log-Wahrscheinlichkeit

Diese Methode nutzt die negativen logarithmischen Wahrscheinlichkeiten der vorhergesagten Tokens (Logprobs) auf Satzebene und betrachtet das Token mit der höchsten negativen Log-

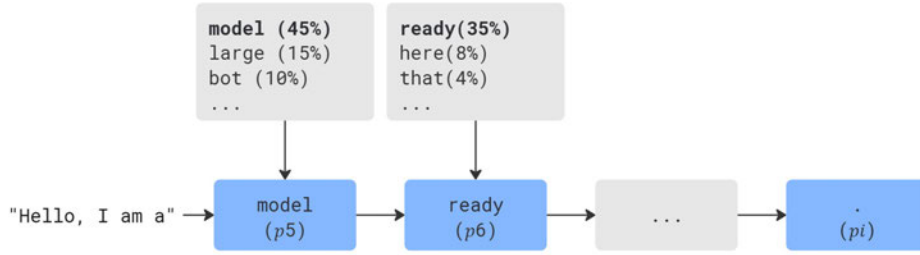


Abbildung 2.7.: In diesem Beispiel der autoregressiven Tokenerzeugung sind die drei Tokenkandidaten mit den höchsten Wahrscheinlichkeiten p_i für die Tokens „model“ und „ready“ an den Positionen $i = 5$ und $i = 6$ zu sehen, welche nach der Greedy Decoding Strategie ausgewählt wurden.

Wahrscheinlichkeit an der Stelle i eines Satzes:

$$\text{Max}(-\log p) = \max_i(-\log p_i) \quad (2.5)$$

Die Bezeichnung „max“ ist auf den ersten Blick etwas irreführend, da sie den Eindruck weckt, als betreffe es das Token mit der höchsten Wahrscheinlichkeit, jedoch ist das genaue Gegenteil der Fall. Überführt man den logarithmischen Wert durch Formel 2.3 in eine prozentuale Wahrscheinlichkeit, so entspricht das Token mit dem höchsten negativen Logprob letztlich dem mit der geringsten Wahrscheinlichkeit. Ein hoher Wert dieses Maßes (niedrige Token-Wahrscheinlichkeit) weist auf geringe Konfidenz bzw. hohe Unsicherheit hin, während eine hohe Wahrscheinlichkeit auf eine hohe Konfidenz hin deutet (Manakul et al., 2023, Y. Huang et al., 2025).

Entropie

Die Entropie misst die Streuung der vorhergesagten Wahrscheinlichkeitsverteilung an einer Position über alle möglichen Tokens des Vokabulars. Hohe Entropiewerte bedeuten, dass eine flache Wahrscheinlichkeitsverteilung vorliegt, die Tokenvorhersagen des Modells verteilen sich also auf viele Tokens. Dies deutet auf größere Unsicherheit und damit auf eine geringere Konfidenz hin (Y. Huang et al., 2025, Y. Yang et al., 2025). Die Entropie an der Position i wird durch die Formel

$$H_i = - \sum_{\tilde{w} \in W} p_i(\tilde{w}) \log p_i(\tilde{w}) \quad (2.6)$$

berechnet. Dabei ist $p_i(\tilde{w})$ die Wahrscheinlichkeit jedes möglichen Tokens an der Position i und W die Menge aller möglichen Tokens aus dem Vokabular (Manakul et al., 2023). Die Entropie wird an jeder Position über den gesamten Vokabularraum des LLMs berechnet.

Veranschaulicht werden kann dies durch das Beispiel in Abbildung 2.7. Möchte man die Entropie an der Position $i = 5$ des Satzes „Hello, I am a model ready to help.“ ermitteln, so multipliziert man für jedes Token die Wahrscheinlichkeit p_5 mit seiner logarithmischen Wahrscheinlichkeit $\log p_5$ und wiederholt dies für alle möglichen Tokens des Vokabulars, wie „model“, „large“, „bot“ und so weiter. Die negative Summe entspricht der Entropie an dieser Position. Die Summe aller Wahrscheinlichkeiten p muss immer 1 ergeben.

Perplexity

Perplexity ist eine etablierte Metrik, die verwendet wird um die Qualität und Genauigkeit von generiertem Text von LLMs zu bewerten (H.-Y. Huang et al., 2024). Die Qualität ist dabei ein Maß für den Zusammenhang und die Vorhersagbarkeit des generierten Texts. Vereinfacht gesagt, kann man sich Perplexity als Unsicherheitsmetrik vorstellen, die beschreibt, wie überrascht das Modell bei der Evaluierung einer Token-Sequenz ist (Mora-Cross und Calderon-Ramirez, 2024). Sie wird durch die folgende Formel beschrieben, die sie als Exponentialwert des Mittelwerts der negativen Log-Wahrscheinlichkeiten eines Satzes definiert (Fadeeva et al., 2024).

$$\text{Perplexity}(p) = \exp\left(-\frac{1}{I} \sum_j \log p_{ij}\right) \quad (2.7)$$

Wie auch in Formel 2.4, wird der Durchschnitt der logarithmischen Token-Wahrscheinlichkeiten über alle Positionen $i \in I$ berechnet, wobei I der Anzahl der Tokens des Satzes entspricht. Exponentiert ergibt sich daraus das Ergebnis der Perplexity. Ähnlich wie auch bei der Entropie, deutet ein höherer Perplexitywert, auf größere Unsicherheit hin, da sich die Tokenwahrscheinlichkeiten auf viele Tokens aufteilen (H.-Y. Huang et al., 2024).

2.3.3. Verbalisierte Unsicherheitsquantifizierung

Eine weitere Art die Unsicherheit von LLMs bei ihren generierten Antworten zu untersuchen, ist das LLM selbst bewerten zu lassen, wie sicher es sich über seine generierte Antwort ist. Dieses Konzept nennt sich „Verbalized Uncertainty“ und wurde erstmals von Lin et al., 2022 vorgestellt. Es funktioniert, indem Large Language Models nicht nur auf eine Frage oder Aufgabe antworten, sondern zusätzlich ihre Unsicherheit in Form von natürlicher Sprache oder einem numerischen Wert bewerten. Eine numerische Angabe, auch „Verbalized Numbers“ genannt, kann Werte zwischen 0 und 1 oder prozentuale Werte zwischen 0 und 100 annehmen, wobei höhere Werte für eine höhere Sicherheit und niedrige Werte für geringe Sicherheit beziehungsweise hohe Unsicherheit stehen. Alternativ kann stattdessen natürliche Sprache („Verbalized Words“) verwendet werden, wobei die Unsicherheit in folgende fünf Stufen

eingeteilt wird: „lowest“, „low“, „medium“, „high“ und „highest“. Jede Stufe entspricht einem Wahrscheinlichkeitsintervall von 0.2, wodurch diese Stufen wiederum direkt einem numerischen Wert zugewiesen werden können. Ein Vorteil gegenüber den in 2.3.2 erwähnten tokenbasierten Ansätzen ist, dass kein Zugriff auf interne Zustände nötig ist und verbale UQ bei jedem Modell funktioniert. Zudem wird angegeben, dass verbalisierte Unsicherheit ein besserer Indikator für epistemische Unsicherheit ist (Tian et al., 2023), da Unsicherheit durch Tokenwahrscheinlichkeiten nicht nur auf mangelndes Wissen hindeuten kann, sondern auch durch die Tatsache entsteht, dass es oft verschiedene Arten geben kann, eine Antwort zu formulieren, wodurch entsprechende Tokenwahrscheinlichkeiten sinken (Lin et al., 2022).

P(True)

Ein anderer Vorschlag der verbalen Unsicherheitsquantifizierung nennt sich „P(True)“. Dieser Hybridansatz wurde von Kadavath et al., 2022 in ihrer Arbeit „Language models (mostly) know what they know“ vorgestellt, wobei Sprachmodelle die Korrektheit ihrer zuvor gegebenen Antwort bewerten müssen. Zunächst wird das Modell aufgefordert, eine Antwort auf eine Frage zu generieren. Um die Unsicherheit zu messen, wird dem Modell die ursprüngliche Frage zusammen mit der soeben generierten Antwort (sampled answer) erneut vorlegt. Nun wird das Modell gefragt, ob die vorgeschlagene Antwort wahr oder falsch ist. Die Quantifizierung der Unsicherheit erfolgt letztlich durch die Wahrscheinlichkeit, welche das Modell dem Token „True“ zuweist.

2.3.4. Konsistenzbasierte Unsicherheitsquantifizierung

Konsistenz kann ebenfalls als Indikator für Unsicherheit dienen. Erzeugt man mehrmals Antworten beziehungsweise Stichproben (Samples) zu derselben Frage, so ist die Konsistenz der Antworten ein Merkmal für die Unsicherheit des Sprachmodells. Ist die Antwort bei allen Iterationen semantisch ähnlich, so deutet dies auf eine hohe Sicherheit hin, weichen die Antworten semantisch stark voneinander ab, so deutet dies auf eine hohe Unsicherheit hin (Y. Yang et al., 2025). Diese Idee wurde auch von Manakul et al., 2023 in ihrer Arbeit „Self-CheckGPT:Zero-Resource Black-Box Hallucination Detection for Generative Large Language Models“ aufgegriffen, um Halluzinationen und faktisch fehlerhafte Antworten zu erkennen. Manakul gibt an, dass voneinander abweichende oder sich widersprechende, zufällig erzeugte Antworten ein Indiz für halluzinierte Fakten sind.

Kosinusähnlichkeit und BERTScore

Bevor die eigentlichen Metriken vorgestellt werden, sollte an dieser Stelle noch darauf eingegangen werden, wie natürliche Sprache mathematisch überhaupt auf Ähnlichkeit überprüft werden kann. Dazu vergegenwärtigen wir uns erneut die in Abbildung 2.2 veranschaulichte Vektorpräsentation im Wortraum. Während die Abbildung zeigt, dass semantische Relationen als Vektoroffset dargestellt werden können, lässt sich die Ähnlichkeit zweier Tokens vergleichen, indem man den Winkel zwischen den zwei entsprechenden Tokenembeddings bestimmt und dabei ihre Richtung untersucht (T. Zhang et al., 2020). Mathematisch abgebildet wird die Kosinusähnlichkeit (cosine similarity) zwischen zwei Tokens A und B wie folgt:

$$\text{cosine}(A, B) = \frac{A \cdot B}{\|A\| \times \|B\|} \quad (2.8)$$

$A \cdot B$ bezeichnen das Skalarprodukt der Tokenvektoren A und B . $\|A\|$ ist die Länge des Tokenvektors A , gleiches gilt für $\|B\|$.

Der BERTScore wurde von T. Zhang et al., 2020 vorgestellt und ist eine gängige Metrik zur Untersuchung der semantischen Ähnlichkeit zwischen zwei Sätzen. Die Berechnung besteht aus mehreren Schritten. Als Erstes werden der Zielsatz und der Referenzsatz in „Contextual Embeddings“ umgewandelt, wobei die Vektorrepräsentation auch die Bedeutung des Wortes in Abhängigkeit des Satzes abbildet. Daraufhin werden die Kosinusähnlichkeiten zwischen jedem Token des Referenzsatzes und jedem Token des Zielsatzes berechnet. Jedes Token wird dabei dem ähnlichsten Token des anderen Satzes zugeordnet. Auf diese Weise werden die Kosinusähnlichkeiten der am besten passenden Tokenpaare verglichen, anstatt die gleichen Tokenpositionen direkt zu vergleichen (T. Zhang et al., 2020). Der Name stammt daher, dass dafür das Encoder-Sprachmodell „BERT“ von Google (Devlin et al., 2019) verwendet wurde.

Sentence-BERT

Anstatt die Embeddings einzelner Tokens eines Satzes zu vergleichen, können auch semantisch aussagekräftige Embeddings eines ganzen Satzes erzeugt werden. Dazu stellten Reimers und Gurevych, 2019 in einer Veröffentlichung Sentence-BERT (SBERT) vor, was eine Modifikation des oben erwähnten BERT Encoder-Sprachmodells ist. Durch mehrere identische neuronale Netze, die über die gleichen Gewichte verfügen, können Satzembeddings erzeugt werden, deren Vektorrepräsentationen zweier semantisch ähnlicher Sätze sehr nahe bei einander liegen. Diese Satzembeddings können dann durch gängige Methoden, wie der Kosinusähnlichkeit, auf Ähnlichkeit untersucht werden. Zudem ist zu erwähnen, dass es sehr schnell darin ist, ähnliche Sätze in einer großen Datenmenge zu finden. Beispielsweise dauerte die Suche nach dem ähnlichsten Satzpaar in einer Menge von 10.000 Sätzen lediglich 5 Sekunden, was mit BERT allein 65 Stunden gedauert hätte (Reimers und Gurevych, 2019).

SelfCheckGPT mit BERTScore

In ihrer Publikation stellten Manakul et al., 2023 insgesamt fünf Varianten vor, von denen der Fokus hier jedoch auf die Variante mit BERTScore liegen soll. Hierbei erhält man auf eine Frage oder eine Aufgabe von einem LLM eine Antwort R und lässt von dem gleichen LLM weitere N Antworten generieren, sodass die Antwortsamples $\{S^1, S^2, \dots, S^n, \dots, S^N\}$ vorliegen. Daraus berechnet man den Wert $S(i)$ mit $S(i) \in [0.0, 1.0]$ durch

$$S(i) = 1 - \frac{1}{N} \sum_{n=1}^N \max_k (B(r_i, s_k^n)) \quad (2.9)$$

wobei der i -te Satz aus R durch r_i und der k -te Satz im n -ten Sample S^n durch s_k^n gekennzeichnet wird. B bezeichnet den BERTScore zwischen zwei Sätzen. Auf diese Weise wird der durchschnittliche BERTScore des i -ten Satzes berechnet, welcher ein höheres Ergebnis liefert, wenn die zwei Sätze ähnliche Informationen enthalten, wodurch auf eine höhere faktische Korrektheit geschlossen werden kann. Die Metrik wurde durch die Invertierung so gestaltet, dass ein hohes Ergebnis auf eine hohe Halluzinationswahrscheinlichkeit beziehungsweise hohe Unsicherheit hindeutet (Manakul et al., 2023).

Antwortkonsistenz mit Ähnlichkeitsfunktion

Ein sehr ähnlicher Ansatz wird von Y. Yang et al., 2025 vorgestellt, bei dem ebenfalls eine Reihe von m Sampleantworten erzeugt wird, für welche durch folgende Formel ein Konsistenzwert berechnet wird:

$$consistency = \frac{2}{m(m-1)} \sum_{i=1}^{m-1} \sum_{j=i+1}^m \text{sim}(r_i, r_j) \quad (2.10)$$

Dabei sei $\{r_1, r_2, \dots, r_m\}$ die Reihe erzeugter Antworten und sim die Ähnlichkeitsfunktion. Yang et al. betrachten in ihrer Arbeit eine exakte Übereinstimmung, grundsätzlich können jedoch auch andere Ähnlichkeitsfunktionen, wie die Kosinusähnlichkeit oder der BERTScore, verwendet werden (Y. Yang et al., 2025).

3. Aufbau und Architektur der Anwendung

Dieses Kapitel soll einen Einblick in die funktionale sowie technische Umsetzung und Architektur der entwickelten Automatisierungsanwendung geben. Des Weiteren wird die Applikation „TSC-Online“ vorgestellt, anhand der die Interaktionsszenarien für die spätere Unsicherheitserhebung durchgeführt werden.

3.1. Beispielapplikation TSC-Online

Die Webanwendung TSC-Online (Technisches Service Center Online) ist eine Prozessmanagementlösung im Bereich der Werkstattausrüstung. Sie wurde von der Absolute Software GmbH für die Unternehmen „Stahlgruber GmbH“ und „PV Automotive GmbH“ entwickelt und befindet sich seit November 2023 als Produktivsystem im Betrieb.

TSC-Online ist eine umfassende Softwarelösung zur Unterstützung und Abwicklung zentraler Geschäftsprozesse im Umfeld der Werkstattausrüstung. Die Anwendung ermöglicht die strukturierte Erfassung und Bearbeitung von Kundenanliegen in Form von Tickets, sowie die Erstellung von Angeboten und Kostenvoranschlägen. Darüber hinaus unterstützt das System die Planung, Disposition und Koordination von Außendienstpersonal.

Die Anwendung ist Teil einer komplexen Systemarchitektur und an verschiedene Schnittstellen angebunden, durch welche zum Beispiel tagesaktuelle Preise und Verfügbarkeiten abgerufen werden und eine automatisierte Dokumenterstellung mit Emailversand möglich ist. Der Themenbereich der Angebote ist in Abbildung 3.1 exemplarisch abgebildet. Ein weiterer zentraler Bestandteil der Anwendung ist die Verwaltung des Kundeninventars. Dies umfasst insbesondere Werkstattausrüstungen, wie Hebebühnen, Diagnosegeräte sowie weitere technische Anlagen. Für jedes Inventar werden detaillierte Informationen, einschließlich Einsatz- und Wartungshistorien erfasst und verwaltet. Anstehende Wartungstermine werden systemseitig erkannt und entsprechende Wartungsvorgänge automatisch generiert. Die Anwendung bietet eine direkte Schnittstelle zwischen Kunden und Mitarbeitern. Kunden können unter einem personalisierten Zugang ihre eigenen Prozesse und Inventare einsehen

3. Aufbau und Architektur der Anwendung

TSC-Online (STA-STAGE) Meine Funktionen Reporting Administratoren project2web Log out

Inr Hauptmenü << ANGEBOT BEARBEITEN

ANGEBOT BEARBEITEN
Erfasst am 09.02.2026 10:04:34 von System TSC-Online Geändert am 13.02.2026 16:45:00 von System TSC-Online

Allgemein

Prozessnummer: OM-1208127-OF

Kundennummer: 747003 Abweichende Kontakt E-Mail: p2w@absolute.de

ABSOLUTE SOFTWARE GMBH (517858)

Ansprechpartner (Freitext): Hendrik Ansprechpartner: Referenz (Kunde): Testreferenz 12345

Infotext des Tickets / Verlauf

Firma **Kunde**

Zahlungskondition **Sonstiges**

Angebotsdetails 1

Titel (intern): Testangebot Absolute Betreff: Testangebot Absolute

Versandschrift (Angebot): Lieferanschrift:

Bindefrist in Tagen: 10 Anfragedatum: 06.02.2026

Angebotsdatum: 06.02.2026

Ändern Löschen Zum Ticket Angebot kopieren E-Mail versenden Letzte Angebotsversion Vorschau Angebot drucken

Abbildung 3.1.: Ein Ausschnitt von TSC-Online anhand eines Testdatensatzes im Angebotsbereich

und bei Bedarf mit zuständigen Ansprechpartnern in Kontakt treten. Für die Einsatzplanung des technischen Personals steht eine interaktive Dispositionsübersicht zur Verfügung. Eine integrierte Kartenansicht unterstützt die effiziente Planung von Anfahrtswegen bei nahegelegenen Aufträgen. Durch die eigenständige Anwendung „SmartClient“ können Techniker mit Disponenten Rücksprache halten und ihre Aufträge direkt mit TSC-Online synchronisieren. Dadurch wird sichergestellt, dass alle relevanten Auftragsinformationen auch offline vor Ort verfügbar sind und bearbeitet werden können.

Zusammengefasst stellt TSC-Online eine Komplettlösung dar, um die gesamte Prozesskette von der Erfassung eines Kundenanliegens über die Angebotserstellung und Ersatzteilbestellung bis hin zur Einsatzplanung und Durchführung und abschließenden Auftragsabrechnung abzuwickeln.

3.2. Entwicklung und Architektur

Das folgende Unterkapitel beschäftigt sich mit der Architektur und Umsetzung der Automatisierungsanwendung. Wie bereits beschrieben, bestand der praktische Impuls dieser Arbeit darin, eine Anwendung zu entwickeln, die in der Lage ist, eine beliebige Webseite innerhalb eines Browsers durch einen Prompt mithilfe eines LLMs zu bedienen. Erste Versuche bestanden darin, eine Webseite lokal zu öffnen und mit Python Bildschirmaufnahmen von der Webseite zu erzeugen. Daraufhin wurden interaktive Objekte und Texte der Bilder mit einer OCR-Bibliothek (Optical Character Recognition) extrahiert und versucht die genauen Bildpunkte einzelner UI-Elemente zu bestimmen. Ein LLM sollte danach eine Aktionssequenz erstellen, worin sie einen Befehl, wie zum Beispiel „click“ zusammen mit einer Bildschirmkoordinate angab. Mithilfe der Python Bibliothek „PyAutoGUI“ konnte dann lokal ein Klick an dieser Bildschirmkoordinate ausgeführt werden. Es zeigte sich jedoch sehr schnell, dass die Erkennung und Extraktion von UI-Elementen sehr unzuverlässig ist, da allein durch ihre Erscheinung nicht immer klar zu erkennen war, ob sie interaktiv sind. Des Weiteren war dieser Ansatz sehr fehleranfällig, da man programmatisch keine Rückmeldung darüber erhielt, ob das korrekte Element, ein anderes Element oder sogar gar kein Element angeklickt wurde.

Daher wurde ein anderer Ansatz gewählt, der mehr Kontrolle über die Objekte einer Webseite bietet und auf deren „Document Object Model“ (DOM) zugreift. Der DOM ist eine strukturierte Darstellung einer Webseite in Form eines Baumes, die von jedem Browser aus dem HTML-Dokument erzeugt wird. Dabei wird jedes HTML-Element, ob Button, Texteingabe oder pure Textanzeige als Node (Knoten) im DOM-Baum dargestellt. Die Baumstruktur bezeichnet die Beziehungen zwischen den Elementen, wie Eltern, Kinder oder Geschwister. Diese Struktur wird auch verwendet, um einzelne Elemente durch ihren sogenannten XPath genau zu identifizieren. XPath steht für „XML Path Language“ und beschreibt den Pfad vom Rootelement, also dem höchsten Element des HTML-Dokuments zu einem Element basierend auf seiner Position im DOM, ähnlich zu Dateipfaden in einem Ordnersystem. Der DOM-Baum wurde in Abbildung 3.2 am Beispiel von TSC-Online veranschaulicht, wobei das Navigationselement für die Seite „Angebote“ hervorgehoben wird.

Um von einer lokalen Pythonanwendung auf eine im Browser laufende Webanwendung zugreifen zu können, wurde das Open-Source Framework „Playwright“ (Microsoft, 2026a) von Microsoft verwendet. Playwright ist ein Werkzeug zum Testen und Automatisieren von Webseiten, indem sie, wie von echten Benutzer*innen gesteuert werden kann. Es arbeitet direkt mit dem DOM einer Webseite und kann Elemente mithilfe von Selektoren, wie dem XPath identifizieren und auswählen. Dadurch ist es möglich, Webseitentests automatisch durchzuführen und auf korrekte Funktionsweise zu überprüfen, wofür es ursprünglich eigentlich entwickelt wurde. Gleichzeitig stellt es die Elemente als Objekte in Python bereit,

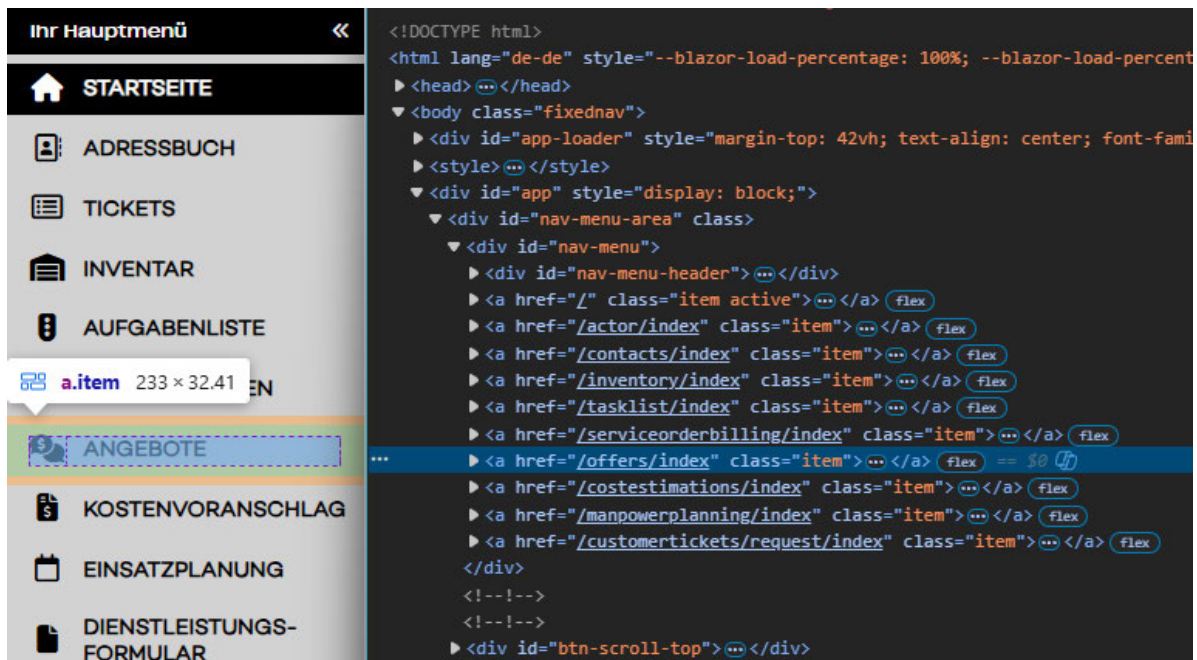


Abbildung 3.2.: Die DOM-Struktur bis zu dem Navigationselement „Angebote“ von TSC-Online

wodurch auch ein direkter Zugriff auf deren Attribute und Zustände, wie zum Beispiel deren CSS-Klassen oder Schreibschutz ermöglicht wird. Vor allem aber können dadurch alle Interaktionsmöglichkeiten, die Nutzer*innen auch haben, aus dem Programm erzielt werden, wodurch eine programmatische Steuerung jeder Webseite möglich ist.

Neben der im Rahmen dieser Arbeit entwickelten Anwendung haben sich inzwischen praxisorientierte Frameworks zur agentischen Browserautomatisierung etabliert, beispielsweise „Browser Use“ (Browser Use, 2026). Im Unterschied zu derartigen Frameworks liegt der Schwerpunkt dieser Arbeit jedoch nicht auf der Bereitstellung eines universell einsetzbaren Webagenten. Vielmehr dient die entwickelte Anwendung als technische und kontrollierbare Basis zur systematischen Untersuchung und Bewertung von Unsicherheitsmetriken.

3.2.1. Funktionale Beschreibung

Die Anwendung ist ein KI-gestützter Automatisierungsagent, dessen Aufgabe es ist, eine Webanwendung in einem Browser über eine grafische Benutzeroberfläche (GUI) zu steuern. In natürlicher Sprache können Nutzer*innen Anweisungen geben oder sagen, was erledigt werden soll und die Anwendung führt die Aufgabe autonom aus. Nach dem Starten der Anwendung erscheint zum einen ein Browserfenster, indem sich eine vordefinierte Startseite öffnet und zum anderen eine einfache Bedienoberfläche, indem es, wie in gängigen

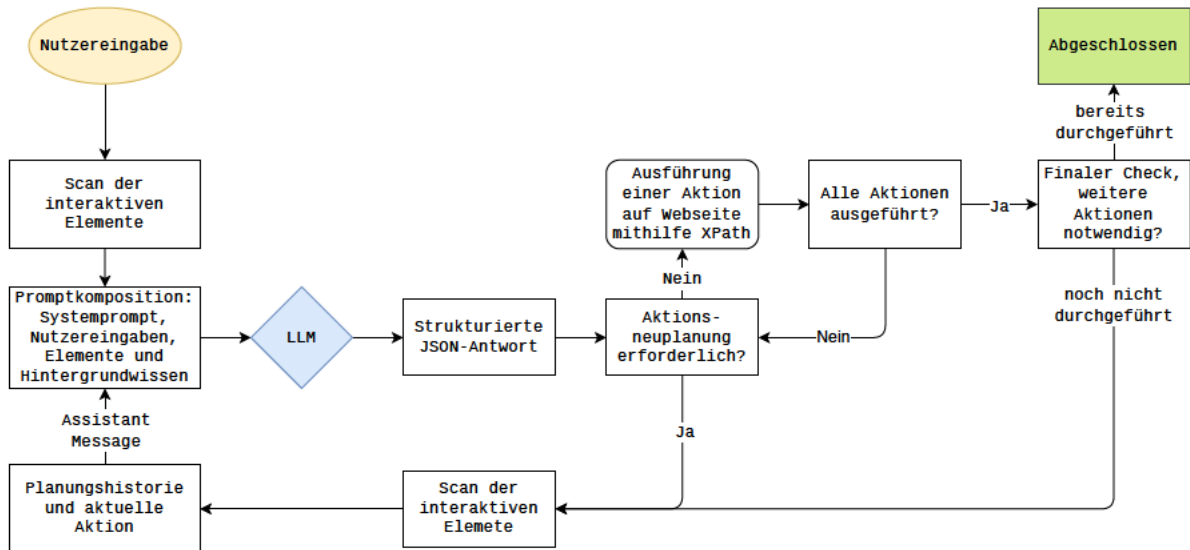


Abbildung 3.3.: Ablaufdiagramm der Webseitensteuerung. Durch eine Nutzereingabe wird die Steuerung initiiert. Interaktive Elemente der Webseite werden extrahiert und zusammen mit der Nutzereingabe und einem Systemprompt an ein LLM gesendet. Dieses antwortet mit einer strukturierten Aktionssequenz. Aktionen werden ausgeführt und die Sequenz gegebenenfalls überarbeitet, bis die Aufgabe als abgeschlossen angesehen wird.

Chatanwendungen, ein zentrales Eingabefeld und einen Chatverlauf gibt. Hier kann eine Aufgabenbeschreibung eingegeben werden, wie zum Beispiel: „Suche nach dem Kunden 'Absolute Software GmbH' und öffne seine Detailseite“. Der Assistent analysiert die Webseite, erstellt einen Aktionsplan und beginnt die Seite Schritt für Schritt zu bedienen, so wie es auch ein Mensch tun würde, solange bis er entscheidet, dass die Aufgabe vollständig abgeschlossen ist. Die wesentliche Funktionsweise der Webseitenbedienung ist in Abbildung 3.3 als Ablaufdiagramm zu sehen.

Der Assistent arbeitet intelligent und anpassungsfähig. Das heißt, nach jeder Aktion wird evaluiert, ob weitere Aktionen ausgeführt werden können oder die Aktionssequenz angepasst und eine Neuplanung durchgeführt werden muss, weil sich zum Beispiel ein Seitenwechsel ereignet hat oder geplante Elemente nicht mehr existieren. In diesem Fall wird die Webseite erneut betrachtet und alle interaktiven Elemente, wie Buttons, Eingabefelder, Links, Menüs oder auch Textelemente, die auf einen Klick reagieren, extrahiert. Dabei werden Informationen wie der XPath, Text, Art des Elements, Attribute und Zustände wie der Schreibschutz erfasst. Dies wird getan, um den Eingabekontext des LLMs nicht unnötig mit zu vielen Informationen zu überladen. Außerdem wird die Aufmerksamkeit des LLMs dadurch auf die wesentlichen Elemente gelenkt.

In dem Fall der Neuplanung wird die Historie der bisher ausgeführten Aktionen dem Ein-

gabeprompt angehängen. Basierend auf den aktualisierten Informationen und den aktuell sichtbaren Elementen wird das LLM aufgefordert die Aktionssequenz zu überarbeiten, so dass es sich stetig den nächsten logischen Schritt überlegt, um dem Ziel näherzukommen. Der Assistent hat ebenfalls die Möglichkeit, Fragen zu stellen und weitere Informationen und Instruktionen anzufordern, wenn Bedarf besteht. Diese Funktionsschleife wird so lange wiederholt, bis es am Ende angibt, dass alle Aktionen erfolgreich durchgeführt und das Ziel erreicht wurde.

3.2.2. Technische Beschreibung

Im folgenden Abschnitt sollen die wichtigsten Kernkomponenten der Automatisierung und ihre Beziehungen zueinander technisch näher beschrieben werden. Das Sequenzdiagramm in Abbildung 3.4 stellt die Prozesskette visuell dar. Den Einstieg bildet die grafische Benutzeroberfläche mit der Komponente `ControlGui`, für welche die Standard-GUI-Bibliothek `Tkinter` verwendet wurde. Die Startseite zeigt ein Eingabefeld für Nutzereingaben, ein Dropdown-Feld für vordefinierte Eingaben, die für die Testszenarien verwendet werden, sowie ein größeres Textfeld, welches die Chathistorie zwischen Nutzer und LLM anzeigt. Es gibt zudem eine Taskleiste, über welches in andere Bereiche, wie den Einstellungen navigiert werden kann. Wird über das Eingabefeld und den Send-Button eine Anweisung gesendet, so wird diese in eine Queue eingespeist, welche von der Komponente `webcontrol` ausgelesen wird. Da die Interaktion mit dem Browser sowie externen Schnittstellen, wie der OpenAI-API asynchron, die Verarbeitung der GUI jedoch synchron ausgeführt wird, geschieht die Kommunikation zwischen GUI und Webcontrol über ein Queue-System, wofür das Framework `janus` verwendet wurde. Die Webcontrol-Komponente ist die Schnittstelle zwischen Browser und Programmlogik. Sie startet und steuert eine Browserinstanz (hier Chrome) durch `Playwright` (siehe Abschnitt 3.2) und ist verantwortlich für das Navigieren, Ausführen von Klicks und Texteingaben und das Abrufen von Informationen von der Webseite. Die Webcontrol-Komponente reagiert auf die Benutzeranweisung und ruft die Methode `plan` der Komponente `PlanningAgent` auf.

`PlanningAgent` ist das Herzstück der Anwendung. Es generiert einen detaillierten Prompt aus verschiedenen Informationen und ist die Schnittstelle zu diversen gehosteten Sprachmodellen. In der Methode `plan` wird der Automatisierungsagent initialisiert. Zuerst werden alle interaktiven Elemente auf der aktuell geöffneten Webseite extrahiert. Dies geschieht durch die Methode `find_interactable_elements` in der Klasse `DomInspector`, welche bereits in `WebControl` initialisiert wurde und dort eine Referenz auf die `Playwright`-Page erhielt. `DomInspector` liest die JavaScriptdatei `clickableElements.js` ein und führt den darin enthaltenen Code auf der aktuellen Webseite aus, um eine Liste aller interaktiven Elemente der Webseite zu generieren. Das Objekt des Buttonelements, um in das Administrationsmenü zu navigieren, sehe dann beispielsweise wie folgt aus:

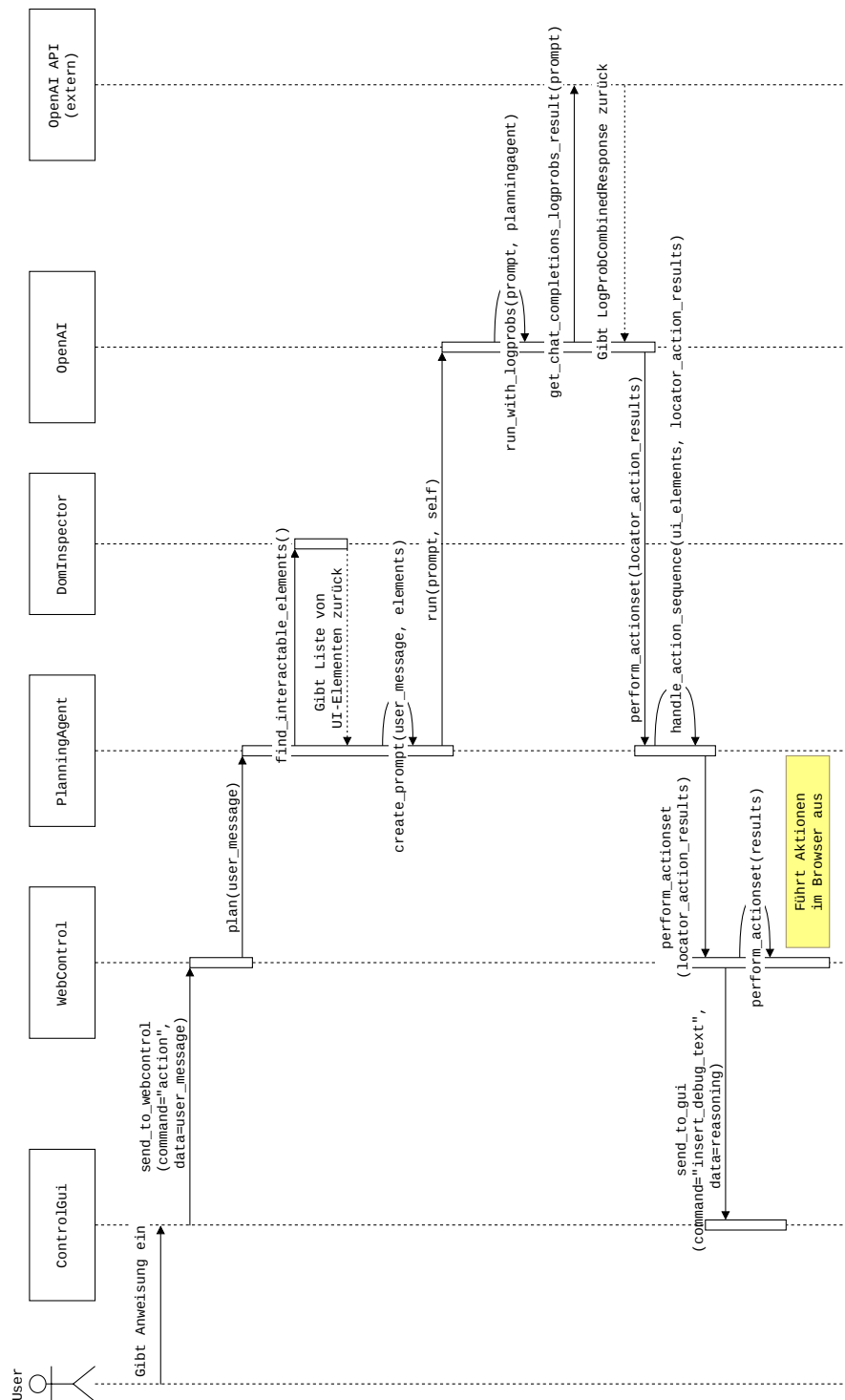


Abbildung 3.4.: Sequenzdiagramm der Automatisierungskette. Die Aufgabe wird durch Nutzereingabe initiiert und führt zu automatisch ausgeführten Aktionen im Browser.

Codeblock 3.1: InteractiveElement-Beispiel

```
1 {  
2     "locatorIterator": 22,  
3     "tag": "BUTTON",  
4     "text": "Administratoren",  
5     "id": "0HNJFJ22GF7AT",  
6     "type": "button",  
7     "xpath": "//*[@id='0HNJFJ22GF7AT']",  
8     "label": "administratoren",  
9     "isDisabled": false,  
10    "attributes": null,  
11    "isVisible": true,  
12    "rowText": ""  
13 }
```

Das Feld `locatorIterator` ist eine automatisch aufsteigend generierte ID, um die Elemente identifizieren zu können, was nicht mit dem Feld `id` zu verwechseln ist. Diese ist eine HTML-ID und da nicht jedes Feld zuverlässig über einen ID-Wert verfügt, wurde das Feld `locatorIterator` eingeführt. Das Feld `isDisabled` beschreibt, ob das Element schreibgeschützt oder deaktiviert ist und `isVisible`, ob das Element verdeckt oder versteckt ist. Zur Veranschaulichung und zu Debugzwecken werden interaktive Elemente auf der Webseite grün umrandet (Abbildung 3.5). Elemente, die deaktiviert oder schreibgeschützt sind, werden rot umrandet dargestellt.

Nun werden alle Informationen in der Methode `create_prompt` im `PlanningAgent` in einen Prompt zusammengeführt. Je nach Fortschritt in der aktuellen Prozesskette wird der Systemprompt leicht modifiziert. Der gesamte Prompt ist in Anhang A.1 zu sehen. So unterscheidet sich der Prompt, je nachdem, ob eine Nutzereingabe initiiert wurde, es sich um eine Anpassung der Aktionssequenz durch eine Neuplanung handelt oder die Aufgabe abgeschlossen wurde. Dem Prompt werden, neben dem Systemprompt, außerdem die Nutzereingabe, Hintergrundwissen, die bisher ausgeführte Aktionssequenz und die interaktiven Elemente der aktuellen Webseite angehängt.

Nun wird in der Methode `ask_llm` die konfigurierte LLM-Implementierung aufgerufen. In diesem Fall wird der vollständig konstruierte Prompt mit `openai.run` an die OpenAI-Komponente übergeben. Diese vermittelt wiederum an `run_with_logprobs`, da wir für die Erhebungen dieser Arbeit, auf die Logprobs der LLM-Antworten zugreifen müssen. In Sektion 4.5 wird näher auf die Implementierung der Datenerhebungen eingegangen, dieses Unterkapitel soll sich jedoch nur mit der Logik der Automatisierung beschäftigen. Für die Kommunikation mit der OpenAI-API wird das `openai`-Modul verwendet und ein `AsyncOpenAI`-Client initialisiert.

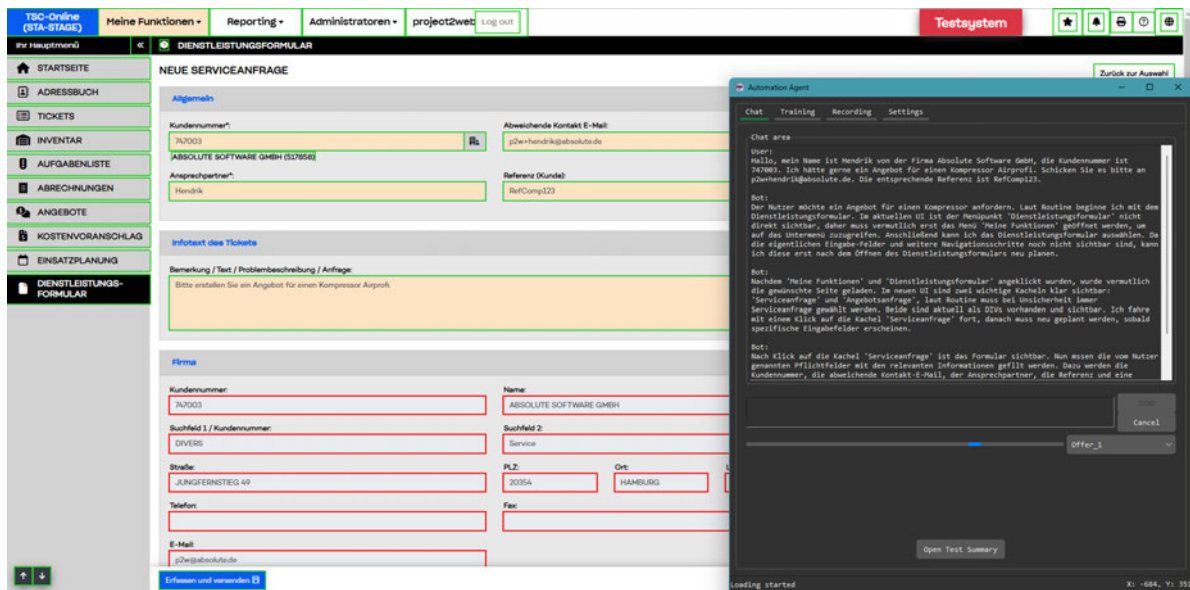


Abbildung 3.5.: Hintergrund: interaktive Elemente werden zur Veranschaulichung grün hervorgehoben, rote sind deaktivierte oder schreibgeschützte interaktive Elemente. Vordergrund: Der Chatverlauf der Automatisierungsanwendung. Bei jeder Sequenz wird das Reasoning des LLMs als Nachricht eingefügt.

Mit der Funktion `get_chat_completions_logprobs_result` rufen wir die OpenAI-API auf und übergeben den Prompt an das Modell „GPT-4.1“, wie der folgende Codeblock zeigt:

Codeblock 3.2: Aufruf der OpenAI-API mit Logprobs

```
1  async def get_chat_completions_logprobs_result(prompt:str, temperature=0.9):
2      completion = await async_client.beta.chat.completions.parse(
3          model="gpt-4.1",
4          messages=prompt,
5          logprobs=true,
6          temperature=temperature,
7          top_logprobs=20,
8          response_format=LogProbCombinedResponse
9      )
10     return completion
```

Dabei setzen wir `logprobs=true` um Zugang zu den internen Logprobs in der Antwort zu erhalten, `temperature=0.9` für ein leicht deterministischeres Tokensampling und `top_logprobs=20` um die maximale Anzahl der von der API angebotenen Top-Tokenlogprobs zu erhalten. Durch den Parameter `response_format` wird das Modell aufgefordert, die Antwort nicht als Freitext, sondern als strukturiertes Objekt der Klasse `LogProbCombinedResponse` zurückzugeben. Dieses

ist im Grunde ein Wrapperobjekt, das neben anderen Feldern ein `LocatorActionResults`-Objekt beinhaltet. Wie das folgende Diagramm zeigt, verfügt es über die Felder `reasoning`, wo die Gedankengänge des LLMs bei der Erzeugung dieser Antwort festgehalten sind, dem boolschen Feld `all_actions_performed`, wo es angibt, dass die Aufgabe vollständig abgeschlossen wurde und dem Array `actionResults`, das die einzelnen auszuführenden Aktionen der Klasse `ActionResult` beinhaltet.

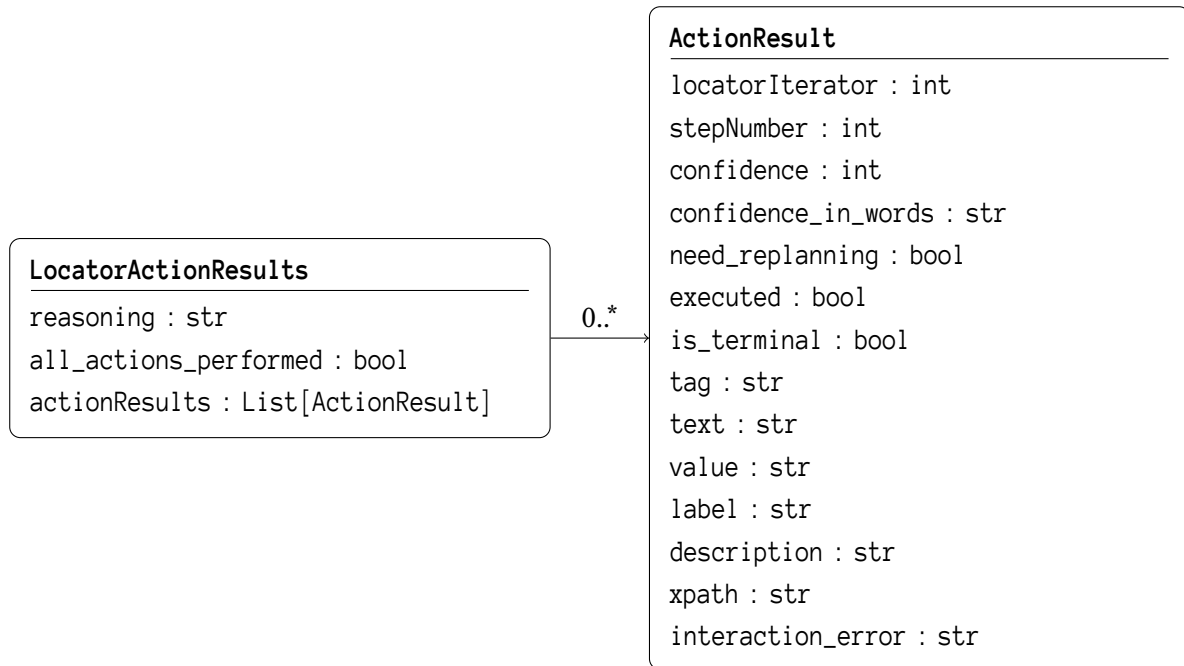


Abbildung 3.6.: Struktur der `LocatorActionResults`-Antwort (JSON) und der enthaltenen `ActionResult`-Objekte

Sofern die LLM-Antwort fehlerfrei in die Klasse `LocatorActionResults` geparsed werden kann, wird die Methode `planningagent.perform_actionset` mit dem Objekt aufgerufen. Hier werden zuerst einige Validierungsschritte durchgeführt. Zum einen wird überprüft, ob die XPath's der Antwortaktionen mit den XPath's der ursprünglich übergebenen Elemente übereinstimmen. Außerdem wird sichergestellt, dass bereits durchgeführte Aktionen nicht versehentlich erneut durchgeführt werden. Nun werden die Aktionen nacheinander durch `webcontrol.perform_actionset` ausgeführt und zuvor der Inhalt des reasonings dem Chatverlauf als Bot-Nachricht über die Queue `send_to_gui` hinzugefügt. Bei jeder Aktion wird versucht das Element anhand seines XPath zu identifizieren und zu finden. Wenn es nicht existiert oder nicht sichtbar ist, wird ein Fehler in das Feld `interaction_error` der Aktion geschrieben und eine Neuplanung veranlasst. Falls sich das Element nicht im derzeitigen Ansichtsfenster befindet, wird es in den Sichtbereich gescrollt. Auf das Element wird zuerst ein Klick ausgeführt. Falls das `value`-Feld über einen Wert verfügt und das Feld beschreibbar ist, wird der Text des `value`-Feldes hineingeschrieben. Sofern es sich bei einer Aktion um den

tag='select' handelt und das value-Feld über einen Wert verfügt, wird nach einem Klick aus dem Select-Element der entsprechende Wert ausgewählt. Wurden alle Aktionen ausgeführt beginnt die Kette von neuem, solange bis die Schleife von dem Nutzer / der Nutzerin unterbrochen wird oder das LLM durch das Feld all_actions_performed angibt, dass die Aufgabe abgeschlossen wurde.

3.2.3. Verwendetes LLM

In dieser Arbeit wurde die OpenAI Python Bibliothek verwendet, um programmatisch aus Python auf die OpenAI-API und dadurch auf die gehosteten Sprachmodelle von OpenAI zuzugreifen. Konkret kam das Modell GPT-4.1 (OpenAI, 2025b) zum Einsatz, welches über den Chat Completions-Endpunkt der OpenAI-API seit dem 14.04.2025 verfügbar ist. GPT-4.1 hat ein Kontextfenster von 1.047.576 Tokens und 32.768 Ausgabetokens (OpenAI, 2025a). Das Kontextfenster beschreibt die maximale Anzahl von Tokens im Eingabeprompt, die das Modell verarbeiten kann. Die genaue Parameteranzahl des Modells wurde nicht bekannt gegeben. Ein wesentlicher Vorteil der OpenAI-API ist, dass sie unter Verwendung des Modells GPT-4.1 den Zugriff auf interne Token-Wahrscheinlichkeiten ermöglicht. Allerdings können für jede Position nur maximal 20 Tokens mit den höchsten Tokenwahrscheinlichkeiten ausgegeben werden. Die meisten öffentlich verfügbaren Sprachmodelle sind jedoch vollständig Closed-Source und liefern keine Informationen über die internen Modellzustände. So bietet auch OpenAI in den neueren GPT-5 Modellen keinen Zugang mehr zu Logprobs.

3.2.4. Entwicklungs- und Systemumgebung

Die Entwicklung, Untersuchungen und Berechnungen wurden auf einem Desktop-PC unter Windows 11 Pro durchgeführt. Die Hardware besteht aus den folgenden Komponenten:

1. **GPU:** NVIDIA GeForce GTX 1080 Ti 11GB
2. **CPU:** Intel Core i7-13700K (3.4GHz)
3. **RAM:** 64GB (DDR4)

Die lokale Hardware wurde nicht für die eigentlichen Modellberechnungen verwendet, diese erfolgte serverseitig durch die OpenAI API, wie in Sektion 3.2.3 beschrieben. Die softwareseitige Entwicklung erfolgte unter Verwendung der IDE (Integrated Development Environment) Visual Studio Code (Microsoft, 2026b). Die Automatisierungsanwendung wurde in Python 3.10.12 entwickelt. Im Wesentlichen wurden die folgenden Python-Bibliotheken verwendet (in alphabetischer Reihenfolge):

1. asyncio (4.8.0)

2. janus (2.0.0)
3. numpy (2.4.2)
4. openai (1.84.0)
5. pandas (3.0.1)
6. playwright (1.51.0)
7. pydantic (2.12.5)
8. scikit-learn (1.7.2)
9. sentence-transformers (5.0.0)
10. tiktoken (0.9.0)

4. Evaluation

Das folgende Kapitel beschäftigt sich mit der Durchführung der Messungen und Erhebung der Messwerte.

4.1. Evaluationsmetrik

Alle Ergebnisse der in diesem Kapitel beschriebenen Messwerte werden durch die Evaluationsmetrik AUROC (Area Under the Receiver Operating Characteristic Curve) verglichen. Sie dient als Bewertung der Unsicherheitsschätzung und misst, wie gut Konfidenzwerte zwischen korrekten und inkorrekten Antworten unterscheiden können (Xiong et al., 2024). AUROC ist ein Maß der Trennschärfe (discriminative power) und wird aufgrund seiner diskriminierenden Fähigkeit häufig neben den Kalibrierungsmetriken¹, wie ECE und Brier Score aufgeführt (Geng et al., 2024). Der Wertebereich liegt zwischen 0 und 1, wobei ein AUROC von 0.5 absolute Zufälligkeit bedeutet und höhere Werte besser sind (Kadavath et al., 2022). Mit dem AUROC kann beispielsweise gezeigt werden, dass Antworten mit gering geschätzter Konfidenz auch mit größerer Wahrscheinlichkeit inkorrekt sind (Ma et al., 2025). Neben dem AUROC wird auch oft die Metrik AUPRC (Area Under the Precision-Recall Curve) zur Bewertung der Fehlererkennung herangezogen.

4.2. Forschungsfrage und Hypothese

Die zentrale Forschungsfrage dieser Arbeit besteht darin, herauszufinden, welche gängigen Unsicherheitsmetriken die zugrundeliegende Unsicherheit von Sprachmodellen bei der Vorhersage und Erzeugung von Aktionsschritten am zuverlässigsten abbildet. Zu diesem Zweck werden ausgewählte Metriken aus drei verschiedenen Kategorien (siehe Kapitel 2.3) eingesetzt und miteinander verglichen. Häufig wurden die Metriken zur Bewertung der Unsicherheit und Konfidenz in anderen Arbeiten anhand von Klassifizierungsaufgaben untersucht, weshalb hier zugleich betrachtet werden soll, ob sie allgemeingültige Werkzeuge sind und sie die

¹Kalibrierung bezeichnet die Übereinstimmung von angegebener Wahrscheinlichkeit einer Vorhersage mit dessen tatsächlichen Genauigkeit (Geng et al., 2024)

Unsicherheit auch innerhalb dieser Domäne abbilden können, wo die Antworten der Sprachmodelle nicht in Form einzelner Worte, sondern komplexer, strukturierter JSON-Objekte auftreten.

Basierend auf den bereits vorliegenden Ergebnissen zur Validität von Unsicherheitsangaben, wird folgendes Hypothesenpaar aus Null- (H_0) und Alternativhypothese (H_1) gebildet:

H_1 : Die Unsicherheitsquantifizierung durch samplingbasierte Konsistenzanalyse erzielt eine höhere AUROC-Trennschärfe als die verbalisierte Unsicherheitsangabe, und diese wiederum eine höhere AUROC als tokenbasierte Unsicherheitsmetriken.

H_0 : Es zeigt sich keine durchgängige Rangordnung, nach der die samplingbasierte Konsistenzanalyse eine höhere AUROC-Trennschärfe erzielt als die verbalisierte Unsicherheitsangabe und diese wiederum eine höhere AUROC-Trennschärfe als tokenbasierte Unsicherheitsmetriken.

Aus den inhaltlichen Hypothesen ergeben sich folgende Vergleichshypothesen, wobei die Unsicherheitsuntersuchung auf Grundlage der Konsistenz mit *consistency*, verbalisierte durch *verbal* und tokenwahrscheinlichkeitsbasierte Methoden mit *tokens* gekennzeichnet sind:

$$H_{1a} : AUROC_{consistency} > AUROC_{verbal} \quad (4.1)$$

$$H_{1b} : AUROC_{verbal} > AUROC_{tokens} \quad (4.2)$$

$$H_{1c} : AUROC_{consistency} > AUROC_{tokens} \quad (4.3)$$

$$H_{0a} : AUROC_{consistency} \leq AUROC_{verbal} \quad (4.4)$$

$$H_{0b} : AUROC_{verbal} \leq AUROC_{tokens} \quad (4.5)$$

$$H_{0c} : AUROC_{consistency} \leq AUROC_{tokens} \quad (4.6)$$

Diverse Quellen geben an, dass die internen Modellzustände, wie die Tokenwahrscheinlichkeiten, gute Indikatoren für die Unsicherheit eines Sprachmodells sind. So erwähnen auch Kadavath et al., 2022, dass korrekt beantwortete Fragen des Modells im Schnitt eine niedrigere Entropie über mögliche Antworten ergaben als solche, bei denen es halluzinierte. Jedoch nehme diese Korrelation bei sehr großen LLMs ab, da sie in der Lage sind vielfältige Formulierungen für korrekte Antworten zu generieren, was die Entropie erhöht, ohne dass Unsicherheit im inhaltlichen Sinne besteht. Dass die Tokenwahrscheinlichkeiten nicht zwangsläufig auf epistemische Unsicherheit hindeuten muss, untermauern Lin et al., 2022. Tian et al., 2023 geben sogar an, dass die Methode der verbalisierten Konfidenz den Kalibrierungsfehler im Vergleich zu tokenbasierten Methoden um 50 % reduziert, jedoch erwähnen sie zugleich, dass ihre Studie sich nur auf Kurzformantworten fokussiere und die Untersuchung von komplexen, längeren Generationen zukünftig noch untersucht werden müsse. Dass auch Methoden, die die Unsicherheit auf Grundlage der Konsistenz untersuchen, besser

abschneiden als tokenbasierte Methoden, welche jedoch starke Indikatoren für Faktizität sein, zeigen Manakul et al., 2023 mit „SelfCheckGPT“. Dieser Aussage schließt sich Kuhn et al., 2022 an. Sie zeigen zudem, dass ihr konsistenzbasierter Ansatz „Semantic Entropy“ im direkten Vergleich mit der von Kadavath et al., 2022 Methode „P(True)“ bessere AUROC Ergebnisse erzielen konnte. Auf dieser Grundlage wurden die oben genannten Hypothesen aufgestellt, welche durch die folgende Methodik beantwortet werden sollen.

4.3. Methodik

In Kapitel 2.3 wurden die theoretischen Grundlagen der Unsicherheitsmetriken bereits detailliert beschrieben. Dieses Unterkapitel soll nun deren Anwendung und Umsetzung in Bezug auf die hier untersuchte Fragestellung behandeln.

Durch einen initialen Prompt wird das Modell aufgefordert, eine Aktionssequenz zu generieren, um die Webseite zu steuern. Während die Schritte ausgeführt werden, muss es immer wieder evaluieren, ob die zuvor generierte Sequenz bearbeitet oder neue Aktionen geplant werden müssen. Mit dem Erhalt der initialen Eingangsinstruktion, antwortet das LLM zuerst mit einer inhaltlichen Themenkategorie innerhalb der Webseite, in die es die erhaltene Instruktion einordnet. Dadurch können die verwendeten Metriken, neben den komplexeren Aktionssequenzen auch in Bezug auf eine einfache Klassifizierungsaufgabe untersucht werden. Insgesamt gibt es sieben verschiedenen Szenarien, welche in Unterkapitel 4.4.1 aufgeführt werden. Die Messungen werden für jedes Szenario einmal ohne und daraufhin mit zusätzlichem Hintergrundwissen durchgeführt, wodurch eine Bewertung der verwendeten Metriken vor allem in Bezug auf die epistemische Unsicherheit ermöglicht wird. Jede Messung wird für jedes Szenario 10-mal wiederholt. Dieser Wert wird von Tian et al., 2023 gestützt, sie untersuchten die Kalibrierung von Modellantworten und stellten fest, dass sie durch mehr als 10 Wiederholungen keine weitere Verbesserung in ihrer Aussagekraft erzielten.

Jeder Aktion fügt das Sprachmodell auch einen Beschreibungstext hinzu (siehe 3.2.2), dessen Tokenwahrscheinlichkeiten analog zu Manakul et al., 2023 auf Satzebene untersucht werden, wie das folgende Unterkapitel beschreibt. Tabelle 4.1 zeigt die Operationalisierung der Unsicherheitsuntersuchung und Messinstrumente.

4.3.1. Tokenwahrscheinlichkeitsbasierte UQ

Die probabilistische Unsicherheit auf Tokenebene wird anhand der durchschnittlichen und minimalen Tokenwahrscheinlichkeit sowie der Entropie der Tokenwahrscheinlichkeiten der Antworten eines LLMs gemessen. Da diese Untersuchungen anhand des Sprachmodells

Tabelle 4.1.: Operationalisierung der zu untersuchenden Unsicherheit den verschiedenen Dimensionen.

Begriff	Dimensionen	Indikatoren	Messinstrument
Unsicherheit	Tokenwahrscheinlichkeitsbasierte Unsicherheit	Logprobs der erzeugten Tokens bei Klassifizierung und Aktionssequenzen	Aggregation der durchschnittlichen und minimalen Tokenwahrscheinlichkeiten und durchschnittlicher und maximaler Entropie
Unsicherheit	Verbalisierte Unsicherheit	Angabe Konfidenz bei Klassifizierung und Aktionssequenzen	Angabe der Konfidenz als numerischen Wert zwischen 0 und 100
Unsicherheit	Konsistenz	Konsistenz der Klassifizierung und Aktionssequenzen	Semantische Ähnlichkeit der Aktionssequenzen bei gleichem Prompt von n erzeugten Samples

GPT-4.1 von OpenAI (siehe Kapitel 3.2.3) durchgeführt werden, können wir genau genommen bei der Betrachtung der ausgegeben Tokenwahrscheinlichkeiten nicht von einem Whitebox-System sprechen, da uns lediglich die top-20 Tokens mit den höchsten Wahrscheinlichkeiten jeder Position ausgegeben werden. Weil wir ansonsten über keinen tieferen Einblick in die internen Zustände verfügen, wird im Folgenden stattdessen die Rede von Greybox-Methoden sein. Diese erfordern ebenfalls Zugriff auf interne Modellinformationen, jedoch nur auf am Ende ausgegebene Tokenwahrscheinlichkeiten (Manakul et al., 2023, Ma et al., 2025). Diese Tokenwahrscheinlichkeiten sollen durch die folgenden Metriken aggregiert werden und der Bewertung der Modellunsicherheit dienen.

1. **Durchschnittliche Tokenwahrscheinlichkeit:** Wie in Abschnitt 2.3.2 beschrieben, betrachtet diese Metrik die durchschnittliche Tokenwahrscheinlichkeit aller Token eines Satzes. Aus dem Ergebnis der logarithmischen Wahrscheinlichkeit aus Formel 2.4 rechnen wir sie in eine prozentuale Wahrscheinlichkeit um und erhalten letztlich folgende Formel:

$$\text{Avg}(p) = \exp\left(\frac{1}{I} \sum_i \log p_i\right) \quad (4.7)$$

Auch hier beschreibt I die Summe aller Tokens im Satz und $\log p_i$ die logarithmische Wahrscheinlichkeit des Tokens an der Stelle i .

2. **Minimale Tokenwahrscheinlichkeit:** Analog zu Abschnitt 2.3.2 betrachten wir hier das Token des Satzes mit der geringsten Wahrscheinlichkeit an der Stelle i . Wir wandeln das logarithmische Ergebnis aus Formel 2.5 durch

$$\text{Min}(p) = \exp\left(-\max_i(-\log p_i)\right) \quad (4.8)$$

in eine prozentuale Wahrscheinlichkeit um.

3. **Entropie:** Die Entropie als Maß für Unsicherheit zu verwenden, wurde bereits in Unterkapitel 2.3.2 erläutert. Normalerweise wird die Entropie über den gesamten Vokabularraum des LLMs berechnet. Da sich die Entwicklung dieser Arbeit jedoch auf die OpenAI-API fokussiert, welche maximal die 20 höchsten Tokenwahrscheinlichkeiten pro Tokenposition ausgibt, wird die Entropie mit $n = 20$ untersucht (analog standen für Manakul et al., 2023 nur die $\text{top}_n = 5$ Tokens bei GPT-3 zur Verfügung) und kann somit nur als approximierte Entropie betrachtet werden. Damit die Summe der verwendeten Tokenwahrscheinlichkeiten pro Position i über alle $n = 20$ Toptokens mit $j \in [1, n]$ wieder 1 ergibt, was für die Entropieberechnung zwingend notwendig ist, muss p_i zuvor normiert werden:

$$|p_i| = \frac{p_i}{\sum_{j=1}^n p_j} \quad (4.9)$$

Daraus ergibt sich für die approximierte Entropieberechnung an der i -ten Position eines Satzes über alle möglichen Tokens j für diese Position:

$$H_{i,20}(p_i) = -\sum_{j=1}^{20} |p_{ij}| \cdot \log(|p_{ij}|) \quad (4.10)$$

Analog zu den oberen beiden Metriken, betrachten wir bei der Entropie ebenfalls die durchschnittliche und maximale Entropie auf Satzebene, sodass wir abschließend zwei Entropie-Metriken erhalten, mit denen wir später die Modellunsicherheit bewerten können (Manakul et al., 2023):

$$\text{Avg}(H) = \frac{1}{J} \sum_j H_{i,20} \quad (4.11)$$

$$\text{Max}(H) = \max_i(H_{i,20}) \quad (4.12)$$

4.3.2. Verbalisierte UQ

Die verbalisierte Unsicherheitsquantifizierung wird durch die Methode der Verbalized Numbers, die erstmals von Lin et al., 2022 vorgestellt und bereits in Unterkapitel 2.3.3 beschrieben

wurde. Sie stellt ein Blackbox-Verfahren dar, wobei das LLM dazu aufgefordert wird jede generierte Aktion innerhalb der Aktionssequenzen mit einem numerischen Konfidenzwert zwischen 0 und 100 zu versehen. Ein Wert von 100 entspricht maximaler Sicherheit. Das Sprachmodell wird bewusst aufgefordert Ganzzahlen zwischen 0 und 100, anstatt von Fließkommazahlen zwischen 0 und 1 zu verwenden, denn die Angabe von Fließkommazahlen resultiert immer in mehr Ausgabetokens, welche aufwendiger zu extrahieren sind. Damit sie mit den anderen Metriken vergleichbar sind, werden die Werte später auf einen Wertebereich zwischen 0 und 1 skaliert.

4.3.3. Konsistenzbasierte UQ

Zur weiteren Bewertung der Unsicherheit in der Kategorie der Blackbox Methoden wird die Konsistenz (siehe Kapitel 2.3.4) der erzeugten Aktionssequenzen und initialen Klassifizierung untersucht. Dazu werden zusätzlich zu jeder vorhergesagten Aktionssequenz 10 weitere Aktionssequenzen generiert, deren semantische Ähnlichkeit gemessen wird. Eine hohe semantische Ähnlichkeit der erzeugten Aktionen deutet auf eine konsistente und zuverlässige Generierung und daher hohe Sicherheit des Sprachmodells hin (Y. Yang et al., 2025). Jede generierte Aktionssequenz kann unterschiedlich viele Aktionen beinhalten, je nachdem, wie granular das Sprachmodell die Sequenz plant oder sie planen kann. Da wir die Unsicherheit nicht für jede Aktionssequenz, sondern für jede einzelne Aktion ermitteln wollen, muss zuerst für jede generierte Aktion die semantisch ähnlichste Aktion in den anderen Sampling-Sequenzen gefunden werden. So wird gewährleistet, dass, auch wenn die Reihenfolge von Aktionen in anderen Sequenzen abweicht, trotzdem ähnliche Aktionen miteinander verglichen werden können, da es in der Regel wichtiger ist, dass die Sequenzen inhaltlich ähnliche Schritte aufweisen, als dass sie in der gleichen Reihenfolge auftreten (man denke beispielsweise an das Ausfüllen eines Formulars).

Nachdem für jede Aktion der vorhergesagten Sequenz die ähnlichsten Aktionen in allen Samples ermittelt wurden, wird die durchschnittliche Kosinusähnlichkeit (siehe 2.3.4) zur Untersuchung der semantischen Ähnlichkeit dieser Aktionen berechnet. Die Kosinusähnlichkeit wird immer zwischen den Satzembeddings der Beschreibungstexte einer jeden Aktion angewandt, welche mithilfe von Sentence-BERT (Abschnitt 2.3.4) erzeugt werden. Die semantische Ähnlichkeit wird bestimmt, indem man die Beschreibungen der Aktionen in den numerischen Vektorraum einbettet und danach ihre Kosinusähnlichkeit betrachtet. Ein höherer Wert steht für größere Ähnlichkeit (Y. Huang et al., 2025). Die Festlegung der erzeugten Samples auf $n = 10$ ist damit zu begründen, dass sich in der Arbeit von Manakul et al., 2023 zeigte, dass sich schon nach unter 10 Samples (im Extremfall genügten sogar vier Samples) keine wesentliche Leistungssteigerung durch weitere Samples mehr erzielen ließ, wobei sie ihre Untersuchungen mit $n = 20$ Samples durchführten. Auch Kuhn et al., 2022 verwendeten in ihrer Arbeit $n = 10$ erzeugte Antworten.

4.4. Versuchsaufbau

Über die Automationssoftware werden Nutzereingaben vorgenommen, aus welchen eine Aktionssequenz erzeugt werden soll, um die Webseite zu steuern (siehe Kapitel 4.4.1). Zunächst wird für die Nutzereingabe eine Kategorisierung des Webseiten-Themenbereichs vorgenommen. Dies bedeutet, dass das Modell für den gegebenen Eingangsprompt (z. B. „Erstelle ein Angebot mit Artikel XY“) den relevanten Themenbereich erkennen soll, hier „Angebote“. Es werden zwei Varianten getestet: einmal ohne und einmal mit aufgabenbezogenem Hintergrundwissen in Form von handbuchähnlichen Routineanweisungen. Die Szenarien variieren in ihrer Art und sollen dazu verwendet werden, das Modellverhalten in Bezug auf die erzeugte Antwortqualität und -sicherheit zu vergleichen.

Für jede Nutzeranfrage generiert das LLM eine mehrschrittige Aktionsabfolge in Form einer JSON-Aktionssequenz. Dabei werden sowohl Greybox- als auch Blackbox-Unsicherheitswerte erfasst.

Tabelle 4.2.: Übersicht der Unsicherheitsmetriken unterteilt in Greybox- und Blackbox-Methoden

Greybox	Blackbox
- Durchschnittliche Tokenwahrscheinlichkeit	- Verbale Selbstangabe der Konfidenz
- Minimale Tokenwahrscheinlichkeit	- Konsistenz mehrerer Aktionssamples
- Durchschnittliche Entropie	
- Maximale Entropie	

4.4.1. Interaktionsszenarien

Nutzereingaben können alles von konkreten Anwendungsanweisungen hin zu Kundenanfragen sein, die ein indirektes Anliegen beinhalten, welches in einen konkreten Aktionsplan umgesetzt werden sollen. Für die experimentelle Untersuchung wurde eine Reihe von typischen Anwendungsfällen aus dem Bereich der webbasierten Geschäftsprozesssteuerung im Kontext der in Kapitel 3.1 beschriebenen Prozessmanagementanwendung „TSC-Online“ definiert. Jeder Anwendungsfall ist dabei als Interaktionsszenario mit einer realen webbasierten Benutzeroberfläche gestaltet und wird durch eine textuelle Nutzereingabe initiiert (z.B. „Erstelle ein Angebot mit Artikel XY“). Die Szenarien bilden häufige Aufgaben im betrieblichen Kontext von TSC-Online ab und setzen sich wie folgt zusammen:

1. **Dienstleistungsformular:** „Hallo, mein Name ist Hendrik von Absolute Software GmbH. Meine Kundennummer ist 747003. Ich habe ein Problem mit meiner Hebebühne, sie klemmt und funktioniert nicht mehr.“

2. **Adressbuch:** „Suche nach dem Kunden Absolute und öffne danach seine Detailseite.“
3. **Angebote:** „Hallo, mein Name ist Hendrik von der Firma Absolute Software GmbH, die Kundennummer ist 747003. Ich hätte gerne ein Angebot für einen Kompressor Airprofi. Schicken Sie es bitte an p2w+hendrik@absolute.de. Die entsprechende Referenz ist RefComp123.“
4. **Inventar:** „Für das Inventar mit der Seriennummer ZYL56182 des Kunden Absolute Software GmbH 747003, aktiviere den Wartungsvertrag ab August 2025 zu einer Wartung nach Vertrag mit jährlichem Wartungsintervall.“
5. **Artikel:** „Wie lautet der Verkaufspreis für den Artikel Geodyna 7750P Radauswuchtmaschine?“
6. **Organisationseinheiten:** „Öffne die Organisationseinheit TSC-302 und setze den Verantwortlichen für die Disposition auf Peter Mustermann.“
7. **Qualifikationen:** „Lösche alle Bremsenprüfungs-Qualifikationen von Peter Mustermann. Lege danach eine Klimageräte-Qualifikation der Stufe Theorie bei ihm an.“

4.4.2. Kontext Hintergrundwissen

Jedes Szenario wurde sowohl ohne als auch mit zusätzlichem Hintergrundwissen untersucht. Dieses wurde dem LLM in Form von Routinen im Prompt mitgeliefert. Für jedes Szenario wurden kurze Informationen angelegt, die die Richtung und den Inhalt grob umreißen. Zusätzlich gibt es ein paar wenige Informationen über die Funktion der Webseite, die für alle Szenarien gleichermaßen gelten. Die angegebenen Routinen können im Anhang [A.2](#) eingesehen werden. Auf diese Weise wird gewährleistet, dass die erhobenen Unsicherheitsmessungen die Unsicherheit aufgrund mangelndem epistemischen Wissens widerspiegelt.

4.5. Implementierung der Datenerhebung

Nachdem in Sektion [3.2.2](#) bereits technisch beschrieben wurde, wie die Prozesskette der Automatisierung umgesetzt wurde, soll dieses Unterkapitel die technische Implementierung der Erhebung der drei einzelnen Metriken behandeln.

4.5.1. Tokenwahrscheinlichkeiten

Um Zugriff zu den Tokenwahrscheinlichkeiten der LLM-Antwort zu erhalten, wird die OpenAI API wie in Codeblock 3.2.2 aufgerufen. Die OpenAI API liefert neben dem eigentlichen Antwortobjekt auch die Tokenwahrscheinlichkeiten als Liste von `ChatCompletionTokenLogprob`-Objekten. Jedes Objekt enthält das generierte Token, dessen Byte-Repräsentation und die zugehörige logarithmische Wahrscheinlichkeit (Abbildung 4.1). Zudem enthält es eine Liste mit den 20 `TokenLogprob`-Objekten mit den höchsten Wahrscheinlichkeiten für diese Tokenposition.

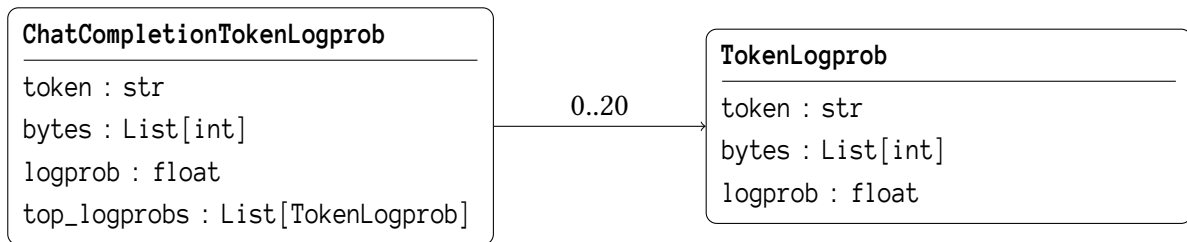


Abbildung 4.1.: Struktur der `ChatCompletionTokenLogprob`-Antwort von der OpenAI API und der enthaltenen `TokenLogprob`-Objekte

Nachdem wir die LLM-Antwort innerhalb der Funktion `run_with_logprobs()` erhalten haben (siehe Unterkapitel 3.2.2), rufen wir die Funktion `logprobs.extract_logprobs()` mit der Antwort auf. Das Modul `logprobs` beinhaltet verschiedene Funktionen, um die Logprobs strukturiert und tokengenau zu extrahieren. Die Schwierigkeit an dieser Stelle ist, dass wir nur an den Logprobs für eine bestimmte Textsequenz, innerhalb der gesamten LLM-Antwort interessiert sind und die Logprobs für diese Positionen zuerst herausfiltern müssen. Denn auch wenn das LLM die Antwort als strukturiertes `LogProbCombinedResponse`-Objekt zurückliefert, so wird es als eine zusammenhängende Stringrepräsentation autoregressiv erzeugt und erst dann in das entsprechende Objekt überführt. Für jede Aktion sollen die Logprobs jedes Tokens des Beschreibungsfeldes `description` ermittelt werden. Lautet der Inhalt des Beschreibungsfeldes beispielsweise „Klicke auf den Button 'Meine Funktionen', um das Hauptmenü zu öffnen.“, wie im folgenden Codeblock 4.1 dargestellt:

Codeblock 4.1: Beispiel der Aktionsbeschreibung

```
'description': 'Klicke auf den Button 'Meine Funktionen', um das Hauptmenü zu öffnen
.'
```

muss diese Zeichenfolge in der Gesamtantwort aller Ausgabetokenlogprobs gefunden werden, damit wir die Logprobs für nur genau diese Sequenz erhalten. Die Logprobs werden als Liste in generierter Reihenfolge jeder Tokenposition ausgegeben (Abbildung 4.1). Stellt man nur die token-Werte nacheinander dar, so sähe ein Ausschnitt beispielsweise aus wie folgt:

```
(...), 'label', '":', 'me', 'ine', 'funkt', 'tionen', '":', 'description', '":', 'K', 'lic', 'ke', 'auf', 'den', 'Button', '":', 'Meine', 'Funktionen', '":', 'um', 'das', 'Haupt', 'men', 'ü', 'zu', 'öffnen', '":', 'xpath', '":', (...)
```

In der Funktion `logprobs.extract_logprobs_for_locatorActionResult()` wird zuerst nach diesem Tokenfenster gesucht. Zu diesem Zweck wird die in Codeblock 4.1 dargestellte Zielsequenz mithilfe der Python-Bibliothek `tiktoken` tokenisiert, wodurch m Tokens resultieren. Dabei kommt derselbe Tokenizer zum Einsatz, der auch vom LLM GPT-4.1 verwendet wird. Auf diese Weise wird sichergestellt, dass die Tokenisierung exakt derjenigen entspricht, die bei der Generierung der Antwort durch die OpenAI API vorgenommen wurde. Sei

$$T = (t_1, t_2, \dots, t_n) \quad (4.13)$$

die Sequenz aller vom Modell generierten Tokens und

$$W = (w_1, w_2, \dots, w_m) \quad (4.14)$$

die gesuchte Tokensequenz (hier `description`). Zur Identifizierung der gesuchten Tokensequenz wird ein gleitendes Fenster der Länge m über die generierte Gesamtliste gelegt. Für jedes Fensterintervall $(t_i, \dots, t_{i+m-1})$ wird die Ähnlichkeit der Fenstersequenz und der Zielsequenz W durch die `SequenceMatcher`-Funktion der Python-Bibliothek `difflib` berechnet, wodurch wir

$$i^* = \arg \max_{1 \leq i \leq n-m+1} \text{sim}(i) \quad (4.15)$$

erhalten. Dadurch erhalten wir den Startindex i^* des Tokenfensters innerhalb der Tokensequenz T , bei dem das Fenster die größte Ähnlichkeit zur gesuchten Tokensequenz W aufweist. Falls es eine genaue Übereinstimmung der tokenisierten Sequenzen gibt, wird diese ausgewählt. Andernfalls wird das Intervall mit der höchsten Ähnlichkeit ausgewählt, sofern ein Ähnlichkeitsschwellenwert von τ nicht unterschritten wird:

$$\text{match}(W, T) = \begin{cases} [i, i + m - 1], & \text{falls } (t_i, \dots, t_{i+m-1}) = W \\ [i^*, i^* + m - 1], & \text{falls } \text{sim}(i^*) \geq \tau \\ \emptyset, & \text{sonst} \end{cases} \quad (4.16)$$

Nachdem das Tokenfenster gefunden wurde, werden die Tokens des Parameternamens `'description'`: herausgefiltert, sodass letztlich nur die Tokens des eigentlichen Inhalts bestehen bleiben. Pro Aktion wird so ein `ActionMetricsResult`-Objekt erfasst, welches die Messwerte der drei verschiedenen Metriken speichert (Abbildung 4.2).

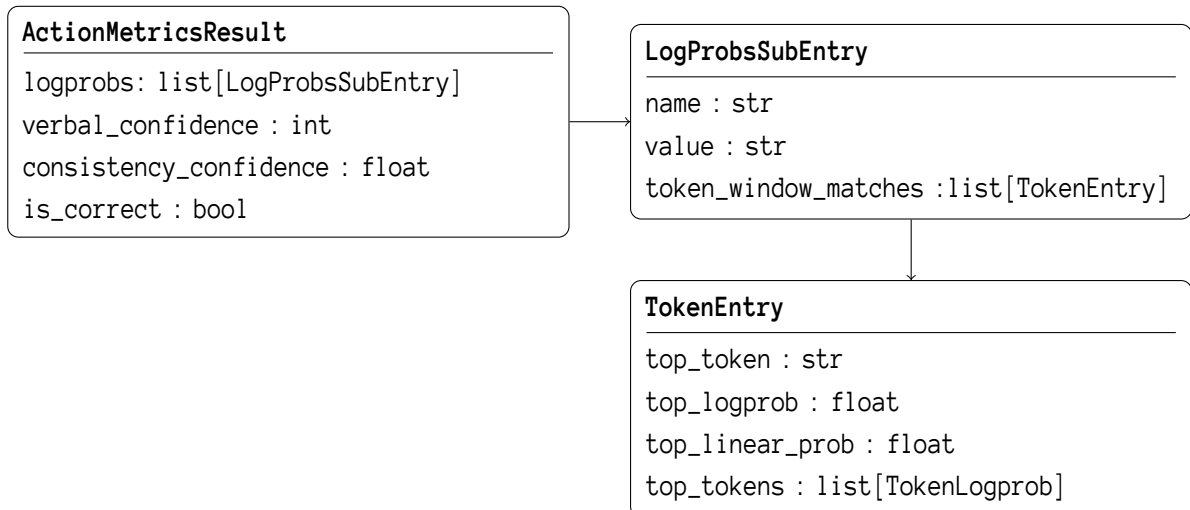


Abbildung 4.2.: Struktur der ActionMetricsResult. Jede Aktion beinhaltet eine Liste von LogProbsSubEntry-Einträgen. Diese enthalten für jeden extrahierten Parameter (in diesem Fall description) die Tokendetails, wie das Token selbst, die Logprobs dieses Tokens sowie die Liste der Toptokens dieser Position.

Der Parameter `verbal_confidence` erfasst das Ergebnis der verbalen Konfidenz aus Unterkapitel 4.5.2 und `consistency_confidence` das Ergebnis der konsistenzbasierten Unsicherheitsuntersuchung aus Unterkapitel 4.5.3. In dem Feld `is_correct` wird festgehalten, ob diese Aktion durch die menschliche Evaluierung als korrekt oder inkorrekt bewertet wurde. Pro gesuchtem Feld der Klasse ActionResult, wird das Tokenfenster in der gesamten LLM-Antwort gesucht und als LogProbSubEntry erfasst. Da für diese Erhebung ausschließlich das Feld `description` untersucht wird, hält die Liste entsprechend immer nur einen Eintrag. Ein LogProbsSubEntry enthält neben dem Namen (hier `description`) und Wert des Feldes (Value) eine Liste der Tokens der gesuchten Tokensequenz. Für jedes dieser Tokens gibt es einen TokenEntry, der wiederum über die logarithmische und lineare Wahrscheinlichkeit, sowie eine Liste der 20 Toptokens verfügt.

4.5.2. Verbalisierte Unsicherheit

Die Messung der verbalisierten Unsicherheit ist vergleichsweise simpel. Hierbei wird das Modell durch den Prompt

Each step must include:
 'confidence': a value between 0 and 100 indicating your confidence that this action is correct and leads to solving the ultimate task

aufgefordert, eine numerische Angabe über seine Konfidenz bei jedem Schritt zu machen (vollständiger Systemprompt siehe A.1). Wie in Diagramm 3.6 gezeigt, enthält jedes Action-Result in der LLM-Antwort dann das Feld `confidence` mit ganzzahligen Werten zwischen 0 und 100. Dieser Wert wird für jedes `ActionMetricsResult` als `verbal_confidence` festgehalten.

4.5.3. Konsistenzbasierte Unsicherheit

Die konsistenzbasierte Unsicherheit wird untersucht, indem mehrere Samples zu der gleichen Frage oder Aufgabe generiert werden. Zu diesem Zweck werden parallel zu dem primären LLM-Aufruf in `run_with_logprobs()` asynchron zehn weitere Ergebnisse erzeugt. Die originale Antwort sowie die zehn Samples werden daraufhin durch die `consistency`-Komponente auf Ähnlichkeit überprüft. Jede Antwort kann unterschiedlich viele Aktionen enthalten. Es kann sein, dass zwei verschiedene Aktionssequenzen zwar die gleichen Aktionen beinhalten, diese sich jedoch nicht zwangsläufig an der gleichen Position innerhalb der Sequenz befinden müssen. Dadurch kann nicht einfach die erste Aktion von Sequenz *a* mit der ersten Aktion von Sequenz *b* verglichen werden, sondern wir müssen zuerst die Aktionen der höchsten Übereinstimmung in jeder Sequenz durch die Funktion `consistency.get_most_similar_entry_consistencies()` suchen.

Die Grundlage dafür bietet die Python-Bibliothek „Sentencetransformer“ (siehe Kapitel 2.3.4). Zu Beginn werden für den Wert des Feldes `description` der originalen und der gesampten Antworten Satzembeddings erzeugt, die Embeddingfunktion kennzeichnen wir im Folgenden als $E(\cdot)$. Für jede originale Aktion einer Sequenz erhalten wir für den Beschreibungstext d die Satzembeddings $D = E(d)$. Gleiches tun wir für die Aktionen der gesampten Antwortsequenzen und erhalten $G_{i,j} = E(g_{i,j})$, wobei $g_{i,j}$ den Beschreibungstext der j -ten Aktion des i -ten Samples bezeichnen. Nun wird für jedes Satzembedding D , das ähnlichste Satzembedding jedes Samples gesucht, indem die Kosinusähnlichkeit berechnet wird. Diese wurde bereits in Kapitel 2.3.4 und Formel 2.8 beschrieben. In Bezug auf den aktuellen Kontext berechnet die Funktion `sim` die Ähnlichkeit durch die Kosinusähnlichkeit mit folgender Funktion:

$$\text{sim}(D, G_{i,j}) = \frac{D \cdot G_{i,j}}{\|D\| \times \|G_{i,j}\|} \quad (4.17)$$

Für jedes Sample wird anschließend die Aktion mit der höchsten Übereinstimmung zur originalen Aktion gesucht.

$$g_i^* = \arg \max_{g_{i,j} \in G_i} \text{sim}(D, G_{i,j}) \quad (4.18)$$

Dadurch wird für jede originale Aktion die semantisch ähnlichste Aktion innerhalb jeder gesampten Sequenz bestimmt.

Nun kann die eigentliche Antwortkonsistenz berechnet werden. Zusammen mit der originalen Antwortbeschreibung übergeben wir die zuvor bestimmten ähnlichsten Antworten an die Funktion `consistency.measure_overall_consistency()`. Für jeden Beschreibungstext werden analog zum vorherigen Abschnitt Satzembeddings durch $E(\cdot)$ erzeugt. Erneut erhalten wir für den Beschreibungstext d der originalen Aktion das Satzembedding $D = E(d)$. Für jede generierten Samplebeschreibung r_j erhalten wir als Satzembedding $R_j = E(r_j)$. Anschließend wird die semantische Ähnlichkeit zwischen der originalen Antwort und jeder gesampleten Antwort mittels Kosinusähnlichkeit bestimmt:

$$\text{sim}(D, R_j) = \frac{D \cdot R_j}{\|D\| \times \|R_j\|} \quad (4.19)$$

Zur Bestimmung der gesamten semantischen Konsistenz der Antworten wird der Durchschnitt aller Ähnlichkeitswerte berechnet:

$$\text{avgsim}(D, R_j) = \frac{1}{n} \sum_{j=1}^n \text{sim}(D, R_j) \quad (4.20)$$

Die Kosinusähnlichkeit liefert Werte im Intervall $[-1, 1]$. Ein Wert von 0 deutet auf keine Ähnlichkeit, ein Wert von -1 sogar auf eine gegengerichtete Bedeutung hin. Da im Rahmen dieser Arbeit nur die semantische Ähnlichkeit von Interesse ist, werden negative Ergebnisse auf 0 gesetzt, wodurch der Ausgabebereich auf $[0, 1]$ transformiert wird. Zur besseren Vergleichbarkeit mit den anderen Unsicherheitsmetriken werden die Ergebnisse mit 100 skaliert. Das Ergebnis wird im Feld `consistency_confidence` des `ActionMetricsResult` (Abbildung 4.2) festgehalten.

5. Ergebnisse

Die Auswertung basiert auf insgesamt 140 Datensätzen, die in Form von CSV-Dateien erfasst wurden. Jeder Datensatz repräsentiert dabei einen vollständigen Durchlauf eines Interaktionsszenarios innerhalb der untersuchten Anwendung und enthält neben der vom LLM erzeugten Aktionen auch die in Kapitel 4 definierten Unsicherheitswerte der verschiedenen Methoden.

Zur Bewertung ihrer Fähigkeit zwischen korrekten und inkorrekten Aktionen zu unterscheiden, wurde die Evaluationsmetrik AUROC verwendet (siehe Kapitel 4.1). Die Berechnung des AUROC erfolgte für jede Metrik separat auf Basis der jeweiligen Unsicherheitswerte und der zugehörigen Ground Truth Label (siehe Anhang A.3). Im Folgenden werden die erfassten Unsicherheitsmetriken systematisch und vergleichbar dargestellt.

5.1. Zusammengefasste Ergebnisse

Die zusammengefassten Ergebnisse sind in Tabelle 5.1 ohne Index dargestellt. Sie beinhalten die Messwerte aller Szenarien, sowohl ohne als auch mit Hintergrundwissen. In der gesamtheitlichen Betrachtung über alle Szenarien hinweg erzielt die konsistenzbasierte Unsicherheitsmetrik den höchsten AUROC-Wert (*consistency* = 0,766). Das zweithöchste Ergebnis erreicht die verbalisierte Unsicherheitsmetrik (*verbal* = 0,735). Unter den tokenbasierten Unsicherheitsmetriken erreicht die maximale Entropie das höchste Ergebnis ($\max(H) = 0,716$), gefolgt von der durchschnittlichen Entropie ($\text{avg}(H) = 0,681$).

Auch in der gesamten Betrachtung zeigt sich, dass die Ergebnisse zwischen den Szenarien variieren. In einigen Szenarien wird eine sehr hohe Trennschärfe erreicht, während sie in anderen Szenarien schwächer ausfällt. Abhängig vom Szenario werden die höchsten Ergebnisse entweder durch die konsistenzbasierte oder verbalisierte Unsicherheitsmetrik erzielt. In Szenario 4 erreicht jedoch die Metrik der maximalen Entropie den höchsten AUROC-Wert ($\max(H) = 0,916$).

Die konsistenzbasierte Unsicherheitsmetrik erreicht über alle Szenarien hinweg die höchsten AUROC-Werte, gefolgt von der verbalisierten Unsicherheit. Die tokenbasierten Metriken

Tabelle 5.1.: AUROC-Werte der untersuchten Unsicherheitsmetriken über alle Test-Szenarien S1 bis S7. Szenarien ohne Hintergrundwissen wurden durch den Index 0, mit Hintergrundwissen mit Index 1 und aggregierte Messwerte ohne Index gekennzeichnet. Die Spalte „Overall“ berücksichtigt die Messwerte aller Szenarien. Jede Reihe beschreibt die AUROC Ergebnisse einer Metrik, wobei verbal die verbalisierten und consistency die konsistenzbasierten Unsicherheitsmetriken bezeichnen. Die Metriken der maximalen und durchschnittlichen Entropie werden durch max(H) und avg(H), der geringsten und durchschnittlichen Tokenwahrscheinlichkeiten durch min(p) und avg(p) dargestellt.

Metrik	S1	S2	S3	S4	S5	S6	S7	Overall
verbal ₀	0,780	1	0,675	0,826	0,591	0,746	0,321	0,724
consistency ₀	0,709	0,979	0,698	0,859	0,527	0,676	0,834	0,741
max(H) ₀	0,693	1	0,660	0,909	0,512	0,606	0,492	0,699
avg(H) ₀	0,767	0,660	0,688	0,748	0,533	0,578	0,464	0,684
min(p) ₀	0,717	0,617	0,575	0,800	0,518	0,517	0,435	0,629
avg(p) ₀	0,757	0,404	0,664	0,716	0,577	0,555	0,432	0,666
verbal ₁	0,961	–	0,587	1	0,798	0,799	0,527	0,687
consistency ₁	1	–	0,837	0,989	0,828	0,813	0,848	0,823
max(H) ₁	0,896	–	0,763	1	0,612	0,823	0,699	0,747
avg(H) ₁	0,792	–	0,739	0,989	0,601	0,951	0,657	0,677
min(p) ₁	0,597	–	0,670	0,989	0,554	0,586	0,616	0,619
avg(p) ₁	0,714	–	0,742	0,966	0,584	0,866	0,636	0,659
verbal	0,830	0,995	0,631	0,858	0,722	0,786	0,440	0,735
consistency	0,692	0,991	0,770	0,886	0,713	0,724	0,843	0,766
max(H)	0,708	0,972	0,713	0,916	0,583	0,664	0,606	0,716
avg(H)	0,739	0,608	0,715	0,745	0,604	0,648	0,572	0,681
min(p)	0,654	0,608	0,622	0,828	0,544	0,548	0,540	0,626
avg(p)	0,721	0,365	0,703	0,733	0,612	0,612	0,546	0,663

weisen ebenfalls eine Trennschärfe oberhalb der Zufallslinie auf, bleiben jedoch in der aggregierten Auswertung hinter den besten Ergebnissen der beiden zuvor genannten Metriken zurück. Abbildung 5.1 zeigt die ROC-Kurven der untersuchten Unsicherheitsmetriken bei der Berücksichtigung aller Messdaten. Auf der x-Achse ist die „False Positive Rate“ aufgetragen, auf der y-Achse die „True Positive Rate“. Die gestrichelte diagonale Linie markiert die Zufallsgrenze. Je weiter eine Kurve oberhalb dieser Linie verläuft und je stärker sie sich der oberen linken Ecke nähert, desto besser trennt eine Metrik zwischen korrekten und inkorrekten Aktionen.

Der durchschnittliche AUROC-Wert liegt bei den Metriken mit Hintergrundwissen bei $AUROC_{\emptyset 1} = 0,779$ und damit über dem durchschnittlichen AUROC ohne Hintergrundwissen ($AUROC_{\emptyset 0} = 0,662$).

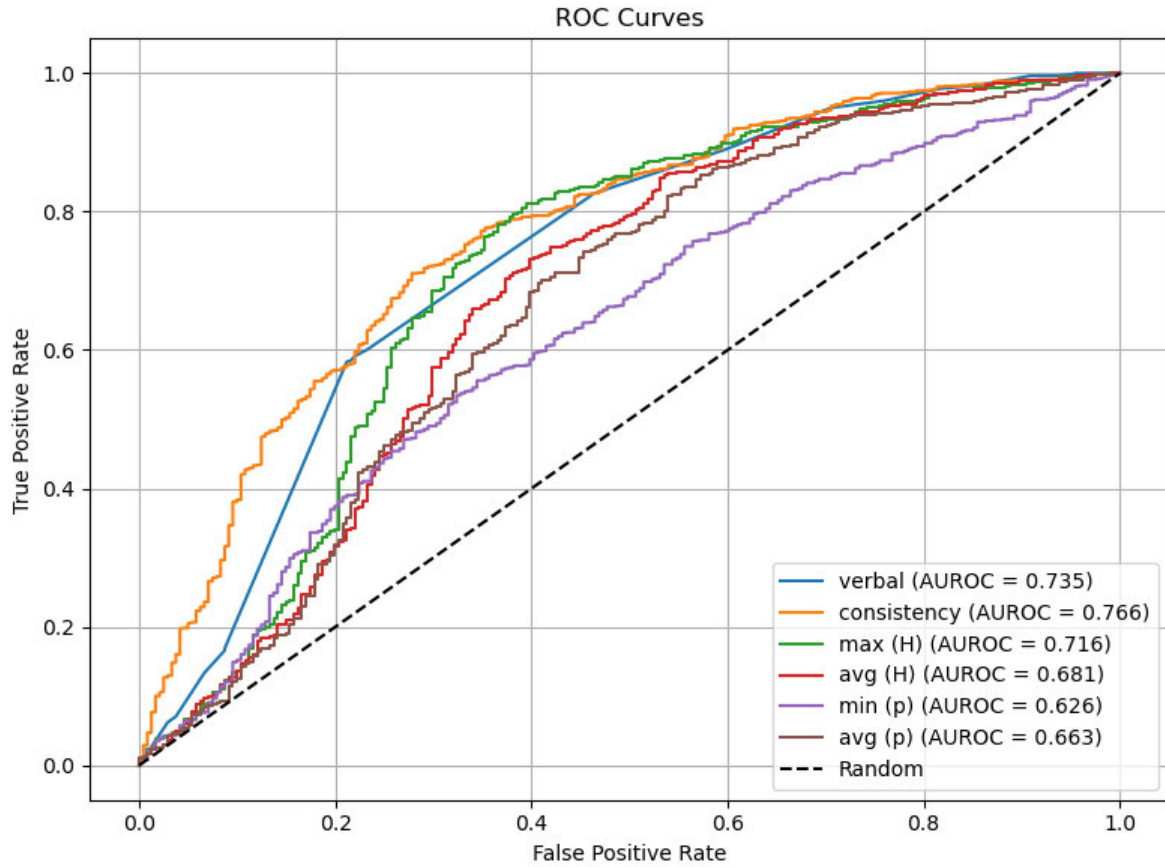


Abbildung 5.1.: ROC-Kurven der zusammengefassten Betrachtung der einzelnen Unsicherheitsmetriken.

5.2. Ergebnisse ohne zusätzliches Hintergrundwissen

Die Ergebnisse ohne zusätzliches Hintergrundwissen wurden in Tabelle 5.1 durch den Index 0 gekennzeichnet. Betrachtet man die Gesamtwerte aller Szenarien dieser Bedingung, erreicht die konsistenzbasierte Unsicherheitsmetrik den höchsten AUROC-Wert ($consistency_0 = 0,741$), gefolgt von der verbalisierten Unsicherheitsmetrik ($verbal_0 = 0,724$). Die Unsicherheitswerte der konsistenzbasierten Metrik stimmen also am ehesten mit den eingetretenen Fällen überein. Die tokenbasierten Unsicherheitsmetriken weisen insgesamt geringere AUROC-Werte auf. Innerhalb dieser Metrikgruppe erzielt die maximale Entropie den höchsten Wert ($max(H)_0 = 0,699$).

Auf Ebene der einzelnen Szenarien zeigt sich, dass die AUROC-Werte nicht in allen Szenarien gleichermaßen ausfallen. Während in einzelnen Szenarien sehr hohe Werte bis 1,0 auftreten, fallen die Ergebnisse in anderen Szenarien deutlich niedriger aus und liegen teilweise sogar unter der Zufallslinie von 0,5. Die verbalisierten und konsistenzbasierten Metriken erzielen abwechselnd die höchsten Ergebnisse, in zwei Szenarien erreicht jedoch auch die maximale Entropie die höchsten AUROC-Werte. In Szenario 7 erreichte die konsistenzbasierte Unsicherheitsmetrik als einzige einen Wert von über 0,5.

Abbildung 5.2 zeigt die ROC-Kurven der untersuchten Unsicherheitsmetriken ohne Hintergrundwissen bei der gleichzeitigen Betrachtung aller Szenarien.

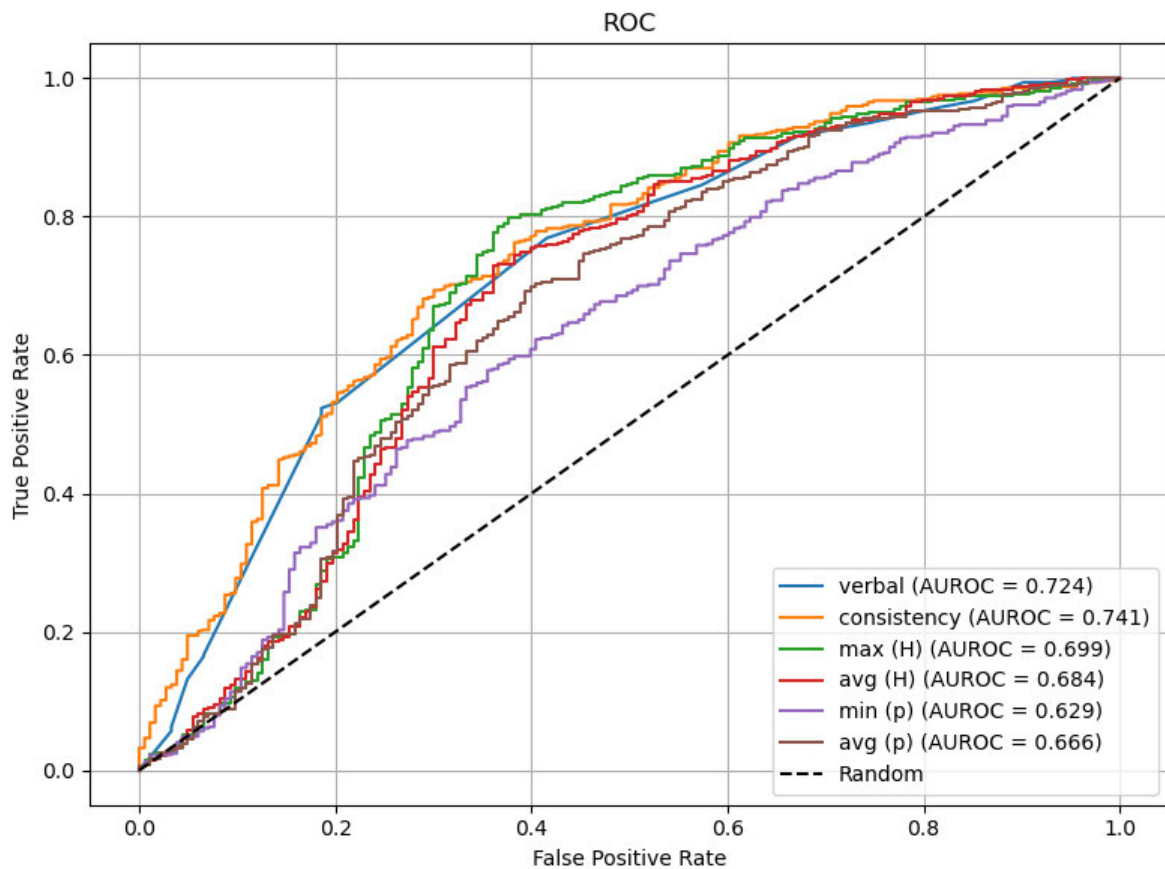


Abbildung 5.2.: ROC-Kurven der einzelnen Unsicherheitsmetriken ohne Hintergrundwissen.

5.3. Ergebnisse mit zusätzlichem Hintergrundwissen

Die Ergebnisse mit zusätzlichem Hintergrundwissen wurden in Tabelle 5.1 mit Index 1 gekennzeichnet. Bei der Betrachtung der Gesamtwerte über alle Szenarien hinweg erreicht

die konsistenzbasierte Unsicherheitsmetrik auch hier die höchste Trennschärfe ($consistency_1 = 0,823$). Darauf folgt die maximale Entropie ($max(H)_1 = 0,747$) als höchste tokenbasierte Metrik, gefolgt von der verbalisierten Unsicherheitsmetrik mit ($verbal_1 = 0,687$). Die weiteren tokenbasierten Metriken weisen insgesamt niedrigere AUROC-Werte auf ($\leq 0,677$).

Betrachtet man die Szenarien einzeln, zeigt sich erneut, dass sich die AUROC-Werte in den Szenarien unterscheiden. Auch in dieser Bedingung mit zusätzlichem Hintergrundwissen werden maximale AUROC-Werte erreicht, jedoch fällt dieses Mal nicht ein einziger Wert unter die Zufallslinie von 0,5. In 4 von 7 Fällen erzielt die konsistenzbasierte Metrik die zuverlässigsten Vorhersagen. Die durchschnittliche Entropie erreicht in Szenario 6 das beste Ergebnis ($avg(H)_1 = 0,916$). Bei der einzelnen Betrachtung von Szenario 2 konnten keine AUROC-Werte berechnet werden, da während der Erhebung keine einzige inkorrekte Aktion auftrat. Die Berechnung des AUROC setzt voraus, dass sowohl korrekte als auch inkorrekte Aktionen im Datensatz vorhanden sind, um die Trennschärfe zu berechnen. Nur dann kann geprüft werden, ob inkorrekte Aktionen höhere Unsicherheitswerte erhalten als korrekte. Abbildung 5.3 zeigt die ROC-Kurven der untersuchten Unsicherheitsmetriken mit

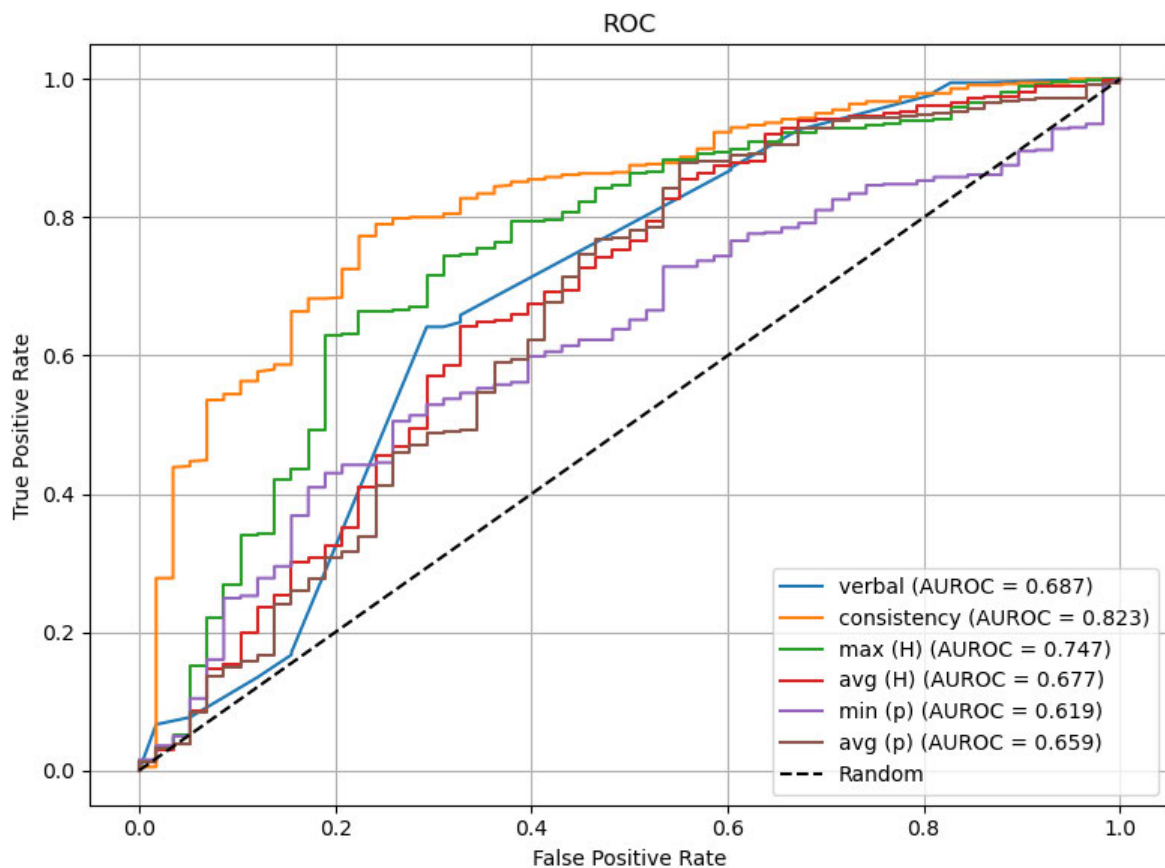


Abbildung 5.3.: ROC-Kurven der einzelnen Unsicherheitsmetriken mit Hintergrundwissen.

Hintergrundwissen bei der Berücksichtigung aller Szenarien.

5.4. Hypothesenprüfung

Zur Prüfung der in Kapitel 4.2 aufgestellten Hypothesen werden die in Tabelle 5.1 dargestellten AUROC-Werte der einzelnen Unsicherheitsmetriken herangezogen. Für die tokenbasierten Unsicherheitsmetriken wird dabei die Metrik mit dem höchsten AUROC-Wert berücksichtigt. Da in allen drei Bedingungen die maximale Entropie die höchsten AUROC-Werte erzielte, wird sie als repräsentative tokenbasierte Metrik verwendet. Die Metriken $\text{avg}(H)$, $\text{min}(p)$ und $\text{avg}(p)$ werden zur weiteren Analyse nicht mehr berücksichtigt.

Die erste Teilhypothese H_{1a} , welche besagt, dass die konsistenzbasierte Unsicherheitsquantifizierung einen höheren AUROC erreicht, als die der verbalisierten Unsicherheitsangabe, wird durch die vorliegenden Ergebnisse gestützt. In der zusammengefassten Betrachtung liegt die konsistenzbasierte Metrik mit $\text{consistency} = 0,766$ über der verbalisierten mit $\text{verbal} = 0,735$. Bei der Betrachtung der Ergebnisse ohne Hintergrundwissen erreicht sie mit $\text{consistency}_0 = 0,741$ ein höheres Ergebnis als $\text{verbal}_0 = 0,724$. Dieser Abstand vergrößert sich bei der Betrachtung mit Hintergrundwissen auf $\text{consistency}_1 = 0,823$ gegenüber $\text{verbal}_1 = 0,687$. Die aufgestellte Rangordnung der Teilhypothese H_{1a} mit $\text{AUROC}_{\text{consistency}} > \text{AUROC}_{\text{verbal}}$ ist damit in allen Bedingungen erfüllt.

Die zweite Teilhypothese H_{1b} besagt, dass die verbalisierte Unsicherheitsquantifizierung einen höheren AUROC-Wert erzielt, als die tokenbasierte Unsicherheitsmetrik. In der zusammengefassten Untersuchung erreicht die verbalisierte Metrik mit $\text{verbal} = 0,735$ einen höheren AUROC-Wert als die tokenbasierte Metrik mit $\text{max}(H) = 0,716$. Ein ähnliches Verhältnis zeigt sich auch bei der Untersuchung ohne Hintergrundwissen mit $\text{verbal}_0 = 0,724$ gegenüber $\text{max}(H) = 0,699$. Betrachtet man die Ergebnisse mit Hintergrundwissen, so erreicht die maximale Entropie mit $\text{max}(H)_1 = 0,747$ jedoch ein höheres Ergebnis als die verbalisierte Unsicherheitsmetrik mit $\text{verbal}_1 = 0,687$. Die Rangordnung $\text{AUROC}_{\text{verbal}} > \text{AUROC}_{\text{tokens}}$ tritt somit nicht in allen drei Bedingungen ein, wodurch die Teilhypothese H_{1b} nur teilweise bestätigt werden kann.

Die dritte Teilhypothese H_{1c} , nach der die konsistenzbasierte Unsicherheitsmetrik ein höheres AUROC-Ergebnis erzielt, als die tokenbasierte Unsicherheitsquantifizierung, wird in allen Bedingungen bestätigt. In der zusammengefassten Betrachtung erreicht die konsistenzbasierte Metrik mit $\text{consistency} = 0,766$ ein höheres Ergebnis als die maximale Entropie mit $\text{max}(H) = 0,716$. Betrachtet man nur die Bedingung ohne Hintergrundwissen, so ergibt sich $\text{consistency}_0 = 0,741 > \text{max}(H) = 0,699$. Mit Hintergrundwissen ergibt sich ein ähnlicher Zusammenhang, sodass die konsistenzbasierte Metrik mit $\text{consistency}_1 = 0,823$ einen höheren

AUROC-Wert erzielt, als die tokenbasierte Metrik mit $\max(H)_1 = 0,747$. Die aufgestellte Rangordnung $AUROC_{consistency} > AUROC_{tokens}$ ist damit in allen drei Bedingungen gegeben.

Betrachtet man die zusammengefassten Messwerte ohne und mit Hintergrundwissen vereint, so bestätigt sich die angenommene Rangordnung $consistency > verbal > tokens$. Granular betrachtet, zeigt sich diese Rangordnung jedoch nicht konsistent über alle Bedingungen hinweg, da in der Bedingung mit zusätzlichem Hintergrundwissen durch die maximale Entropie ein höherer AUROC-Wert erzielt wurde, als durch die verbalisierte Unsicherheitsmetrik. Die entsprechende Nullhypothese H_0 kann daher nur teilweise verworfen werden, da die Teilnullhypothese H_{0b} nicht für alle Bedingungen konsistent widerlegt werden konnte.

5.5. Ergebnisse der Klassifikation

Als Nebenuntersuchung wurde das LLM zu Beginn jeder initiierten Aktionssequenz aufgefordert, die inhaltliche Themenkategorie anzugeben, in die es den Inhalt des Prompts einordnen würde (Siehe Kapitel 4.3). Für jeden durchgeführten Messdurchlauf wurde ein Klassifikationsschritt erfasst, sodass insgesamt 140 Messwerte vorliegen. Abbildung 5.4 zeigt die AUROC-Ergebnisse der untersuchten Unsicherheitsmetriken für die Themenklassifikation. Auch hier erzielt die konsistenzbasierte Unsicherheitsmetrik das höchste Ergebnis mit $consistency_c = 0,963$. Die tokenbasierten Metriken übertreffen in dieser Auswertung die AUROC-Werte der verbalisierten Unsicherheitsmetrik. Die maximale Entropie erreichte mit einem AUROC-Wert von $\max(H)_c = 0,828$ das höchste Ergebnis der tokenbasierten Metriken. Das niedrigste Ergebnis erzielt die verbalisierte Unsicherheitsmetrik mit einem AUROC-Wert von $verbal_c = 0,587$.

Während der Klassifikation sind insgesamt 10 inkorrekte Klassifizierungen aufgetreten. Davon traten alle während des Szenarios 1 ohne Hintergrundwissen auf. Tabelle 5.2 zeigt vorhergesagte Konfidenzwerte für jeden dieser inkorrekten Vorhersagen. Es fällt auf, dass die verbalisierten Vorhersagen konsistent sehr hoch angegeben wurden und in den meisten Fällen sogar Werte von 95 % annehmen.

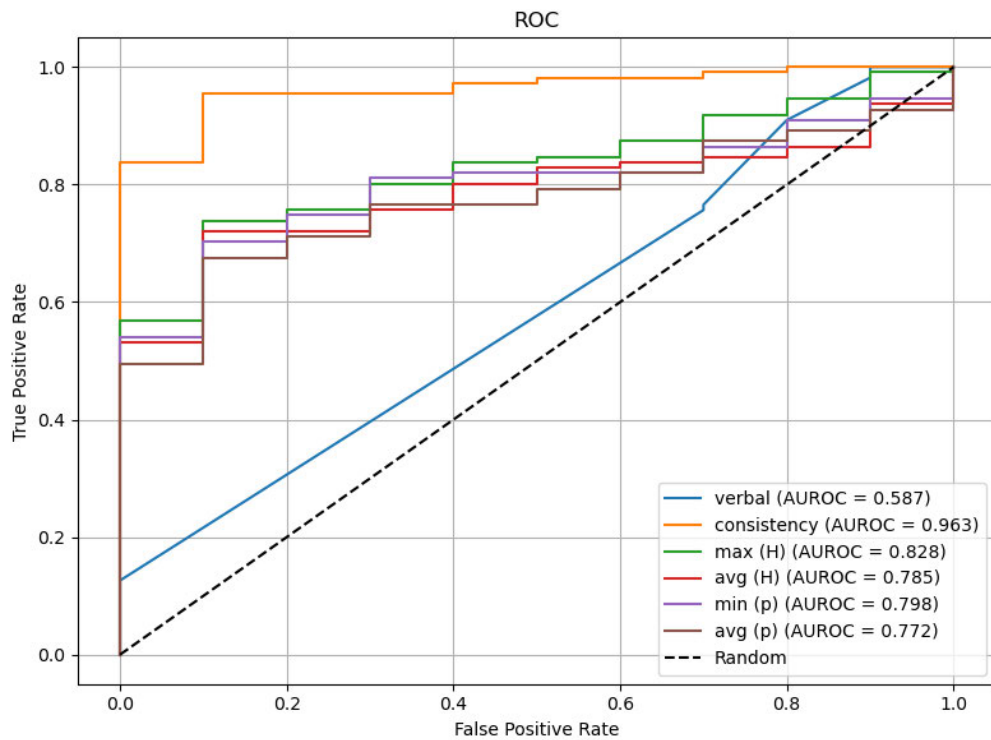


Abbildung 5.4.: ROC-Kurven der zusammengefassten Betrachtung der einzelnen Unsicherheitsmetriken für die Themenklassifikation.

Tabelle 5.2.: Messwerte der untersuchten Unsicherheitsmetriken der 10 aufgetretenen inkorrekten Klassifikationen.

	verbal_c	consistency_c	$\text{max}(H)_c$	$\text{avg}(H)_c$	$\text{max}(p)_c$	$\text{avg}(p)_c$
1	95,00	71,01	71,18	90,39	46,48	82,16
2	95,00	95,17	76,49	92,16	59,73	86,58
3	70,00	41,81	78,27	90,84	64,42	87,99
4	95,00	80,05	77,75	89,55	61,51	84,04
5	95,00	63,05	62,89	87,63	33,65	77,88
6	95,00	80,68	90,74	96,91	93,32	97,77
7	90,00	54,52	77,12	93,59	56,22	88,69
8	95,00	81,24	82,79	94,26	83,74	94,58
9	95,00	36,33	74,21	91,40	28,83	76,28
10	85,00	56,35	81,38	94,43	76,94	93,66

6. Diskussion

Das folgende Kapitel beschäftigt sich mit der Diskussion und Interpretation der in Kapitel 5 dargestellten Ergebnisse. Darüber hinaus werden die Ergebnisse in den Forschungsstand eingeordnet und die Stärken und Limitationen der Methoden ausgeleuchtet. Abschließend wird auf die Bedeutsamkeit der Ergebnisse für praktische Anwendungen eingegangen und ein möglicher Ausblick auf weitere Untersuchungen gegeben.

6.1. Interpretation der Ergebnisse

Die Ergebnisse in Kapitel 5 zeigen, dass die untersuchten Verfahren zur Unsicherheitsquantifizierung im Kontext mehrschrittiger Aktionssequenzen in der Lage sind, zwischen korrekten und inkorrekten Aktionen zu unterscheiden. Dabei ist besonders hervorzuheben, dass die konsistenzbasierte Unsicherheitsmetrik, sowohl mit als auch ohne Hintergrundwissen, den höchsten AUROC-Wert erreicht und damit insgesamt die robusteste Metrik zur Beurteilung der zugrunde liegenden Unsicherheit von Modellantworten darstellt. In der zusammengefassten Betrachtung aller Szenarien erreichte sie einen AUROC von 0,766. Dieser Wert ist so zu interpretieren, dass die Metrik korrekte und inkorrekte Aktionsvorhersagen mit einer Wahrscheinlichkeit von 76,6 % zutreffend voneinander unterscheiden kann. Dies spricht dafür, dass die Konsistenz im Bereich der webbasierten Prozessautomation ein zuverlässiger Indikator für Unsicherheit ist. Die wiederholte Erzeugung von Vorhersagen liefert also stabilere Erkenntnisse über die Zuverlässigkeit einer Modellantwort, als lokale Tokenwahrscheinlichkeiten oder die eigene verbalisierte Einschätzung des LLMs.

Die Ergebnisse zeigen außerdem, dass die aufgestellte Hypothese nur teilweise bestätigt werden konnte. Auf ganzheitlicher Betrachtung konnte die erwartete Überlegenheit der verbalisierten gegenüber tokenbasierter Unsicherheitsmetriken zwar bestätigt werden, unterlag jedoch in der Bedingung mit zusätzlichem Hintergrundwissen der maximalen Entropie. Dies deutet darauf hin, dass verbalisierte Einschätzungen von LLMs definitiv als Indikatoren für Unsicherheit dienen, jedoch stärker vom Kontext abhängig sind als zunächst angenommen. Verbalisierte Konfidenzangaben waren häufig überkonfident und wurden sehr hoch angegeben, obwohl die ausgeführte Aktion nicht korrekt war.

Zwar fallen die tokenbasierten Unsicherheitsmetriken aggregiert hinter die konsistenzbasierten und verbalisierten Metriken, jedoch liegen sie fast immer über der Zufallslinie. Besonders Szenario 4 zeigt mit sehr hohen AUROC-Werten der maximalen Entropie, dass tokenbasierte Verfahren in bestimmten Kontexten sehr stark zwischen korrekten und inkorrekten Vorhersagen trennen können. Tokenbasierte Metriken sind also grundsätzlich ebenfalls nützliche Unsicherheitsindikatoren, jedoch ist ihre Stabilität und Robustheit stärker abhängig vom Kontext.

Darüber hinaus zeigt sich auch, dass die Ergebnisse in den unterschiedlichen Szenarien variieren. Ein Grund dafür kann die Komplexität sein. Szenarien mit kurzen Referenzlösungen führten tendenziell zu stabileren Aktionsvorhersagen. Szenarien mit mehr erforderlichen Einzelschritten führten häufiger zu Fehlern, wenn Teilschritte nicht geplant oder ausgeführt wurden, die jedoch für spätere Aktionen notwendig waren. Dies fällt besonders auf, wenn man die Ergebnisse von Szenario 2, dessen Referenzlösung nur 4 Aktionen umfasste, mit denen von Szenario 3 (13 Referenzaktionen) vergleicht. Ebenso konnte eine falsche Aktion zu fehlerhaften Folgeschritten führen, insbesondere wenn Menüpunkte der Webseite mehrdeutig waren. So fiel es dem LLM trotz Hintergrundwissen nicht immer leicht, zur Angebotserstellung in das Menü „Angebote“ zu navigieren, sondern wählte vereinzelt auch das Menü „Tickets“. Bei der verbalisierten Metrik fiel zudem auf, dass das LLM an fehlerhaften Aktionen länger mit hoher Konfidenz festhielt, als es bei der konsistenzbasierten Metrik der Fall war.

Die deskriptive Statistik in Anhang A.4 zeigt zudem, dass sich die untersuchten Metriken in ihrer Verteilung stark unterscheiden. Betrachtet man den Median der Metriken, so fällt auf, dass die verbalisierten Konfidenzwerte insgesamt sehr hoch ausfallen, jedoch noch mit Abstand unter den tokenbasierten Metriken liegt, welche im Median allesamt Werte von annähernd 100 % annehmen. Die maximale Entropie $\max(H)$ und $\min(p)$ weisen dabei hohe Standardabweichungen auf. Die konsistenzbasierte Metrik zeigt im Gegensatz eine breitere und gleichmäßigere Verteilung ohne eine solche Häufung am Maximum, was ein Grund für ihre robusten und stabilen Vorhersagen sein könnte.

Abschließend ist zu erwähnen, dass alle hier untersuchten Verfahren, ob Konsistenz, selbst verbalisierte Verfahren oder auch die maximale Entropie grundsätzlich deutliche Indikatoren für Modellunsicherheit innerhalb der Domäne von webbasierter Prozessautomation darstellen. Die maximale Entropie weist, stellvertretend für tokenbasierte Methoden, als schlechteste untersuchte Metrik immer noch einen AUROC von 0,716 auf und liegt damit noch immer weit über der Zufallsgrenze. Die untersuchten Metriken lassen sich daher durchaus auf diese Domäne übertragen.

6.2. Einfluss von Hintergrundwissen

Der durchschnittliche AUROC der Bedingung mit zusätzlichem Hintergrundwissen lag mit 0,779 deutlich über dem Durchschnittswert ohne Hintergrundwissen mit 0,662. Mehr Wissen scheint die Fähigkeiten der einzelnen Metriken entscheidend darin zu verbessern, zwischen korrekten und inkorrekten Aktionen unterscheiden zu können. Je mehr das LLM über die gestellte Aufgabe und den Kontext weiß, desto zutreffender fallen auch die Konfidenzvorhersagen aus. Vereinzelt zeigt sich jedoch auch eine Abnahme des AUROC. Die Trennschärfe der verbalisierten Unsicherheitsmetrik liegt in Szenario 3 mit Hintergrundwissen unter der Bedingung ohne Hintergrundwissen. In einzelnen Fällen kann mehr Kontext auch zu einem Bias der Konfidenz führen. Durch den höheren Informationsgehalt, nimmt das LLM an, sicherer darin zu sein, die Aufgabenstellung zu lösen und dadurch zwangsläufig mehr Wissen über alle einzelnen Problemschritte zu haben. Das muss nicht zwangsläufig der Fall sein, denn das zusätzliche Wissen muss nicht unbedingt für alle enthaltenen Aktionen hilfreich sein.

Über alle Szenarien hinweg wurden insgesamt 1803 Aktionen durchgeführt, davon 968 Aktionen ohne zusätzliches Hintergrundwissen (54 %) und 835 Aktionen mit Hintergrundwissen (46 %). Es wird deutlich, dass mehr Wissen über den Themenkontext zu einer zielgerichteteren Aufgabenbewältigung führt. Mit Hintergrundwissen wurden die Aufgaben nach weniger „ausprobierten“ Aktionen erledigt als ohne.

6.3. Einordnung in den Forschungsstand

Die Ergebnisse stimmen mit Arbeiten überein, die konsistenzbasierte Verfahren als besonders robuste Unsicherheitsindikatoren beschreiben (Manakul et al., 2023, Chen und Mueller, 2024). Die in Abschnitt 5.1 zusammengefassten Ergebnisse stützen die Befunde von Y. Yang et al., 2025, wonach besonders konsistenzbasierte Metriken auch bei mangelndem Wissen sehr leistungsfähig bleiben. Dass die Konsistenzmessung effektiver darin ist Fehler vorherzusagen, als verbalisierte Konfidenzen wird zudem von Xiong et al., 2024 untermauert.

Für die verbalisierte Unsicherheitsmetrik ergibt sich ein gemischtes Bild. Einerseits erreichen die verbalisierten Unsicherheitsmethoden in der zusammengefassten Auswertung gute Ergebnisse und übertreffen ohne Hintergrundwissen die tokenbasierten Verfahren. Andererseits zeigen die durchwachsenen Ergebnisse mit Hintergrundwissen und die inkorrekten Fälle während der Klassifikation, dass die Konfidenzen sehr hoch ausfallen, obwohl die Vorhersage falsch ist. Das passt zu Kadavath et al., 2022 und Xiong et al., 2024, die eine deutliche Überkonfidenz bei verbalisierten Angaben beobachteten, oft mit Werten zwischen 80 % und 100 %. Xiong et al., 2024 beschreiben dies als Imitation von menschlichen Mustern. Auch M. Zhang et al., 2024 bestätigten, dass die angegebene Konfidenz und Korrektheit der Antworten stark

abweichen kann. Die Ergebnisse stehen jedoch nicht im Einklang mit den Resultaten von Tian et al., 2023 oder Lin et al., 2022. Tian et al., 2023 zeigten, dass verbalisierte Konfidenzen bei RLHF-Modellen¹ meist besser kalibriert sind, als bedingte Tokenwahrscheinlichkeiten.

Die Beurteilung tokenbasierter Methoden fällt ebenfalls gemischt aus. Auch wenn sie in der Gesamtbetrachtung hinter konsistenzbasierten und verbalisierten Unsicherheitsmetriken zurückbleiben, liegen sie doch fast durchgehend oberhalb der Zufallslinie. Dadurch dienen sie als informative Indikatoren für Unsicherheit. Dabei sticht die maximale Entropie in allen drei Bedingungen als stärkste tokenbasierte Metrik besonders heraus. Das ist in Übereinstimmung mit Fadeeva et al., 2024, die berichten, dass die Aggregation der maximalen Tokenentropien zuverlässigere Ergebnisse als der Durchschnitt oder das Minimum der Tokenwahrscheinlichkeiten liefert und sogar die Perplexity übertrifft. Die variierende Leistung der maximalen Entropie wird hingegen von Kadavath et al., 2022 gestützt. Sie gaben an, dass die Unsicherheitsquantifizierung auf Grundlage der Entropie mit zunehmender Modellgröße stark sinkt, da sie in der Lage sind, eine korrekte Antwort auf vielfältige Weisen zu formulieren. Dies erhöhe die Entropie, ohne dass Unsicherheit im inhaltlichen Sinne besteht. Dieser Ansicht ist auch Lin et al., 2022. Sie beschrieben ihre Erkenntnisse jedoch anhand offener Aufgaben. Die Formulierungsmöglichkeiten in dieser Untersuchung waren insofern gelenkter und eingeschränkter, als dass die Bezeichnungen und Inhalte der interaktiven Elemente übergeben wurden und daher vorgegeben waren. Das LLM musste vielmehr entscheiden, mit welchen Elementen es in welcher Form interagiert, als sich eine Formulierung für die Elemente zu überlegen.

6.4. Stärken, Schwächen und Verzerrungen

Die Untersuchungen dieser Arbeit wurden im Kontext einer konkreten Prozessmanagementanwendung im Produktivbetrieb durchgeführt. Dadurch wurden die Metriken nicht nur an einer abstrakten Klassifikationsaufgabe untersucht, sondern anhand eines angewandten Beispiels aus der realen Welt, wodurch die Ergebnisse besonders praxisnahe Bedeutung haben. Zudem wurden verschiedene Methoden innerhalb derselben Versuchsumgebung verglichen, wodurch ihre Ergebnisse relativ eingeordnet werden können, anstatt sie separat und isoliert zu bewerten. Durch die Betrachtung der Metriken unter verschiedenen Bedingungen (mit und ohne Hintergrundwissen) konnte zudem gezeigt werden, dass der Kontext für die Zuverlässigkeit der Metriken eine Rolle spielt.

¹Reinforcement Learning from Human Feedback (RLHF) ist eine Technik des maschinellen Lernens, wobei ein Modell durch menschliches Feedback und Reinforcement Learning trainiert wird und dadurch menschliche Präferenzen erlernt. Minaee et al., 2025

Eine Schwäche der konsistenzbasierten Methode liegt in ihrem hohen Ressourcenbedarf, was ebenfalls von Ma et al., 2025 kritisiert wurde. Da für jede erzeugte Antwort viele zusätzlich erzeugte Sampleantworten benötigt werden, steigt der Rechenaufwand deutlich, was Kosten und erhöhten Stromverbrauch verursacht. Berücksichtigt man, dass ein durchschnittlicher Prompt mit Googles Gemini laut Elsworth et al., 2025 im Median so viel Strom verbraucht, wie neun Sekunden Fernsehen, wird klar, dass dieser Aspekt nicht zu vernachlässigen ist. Token-basierte Verfahren haben wiederum die Einschränkung, dass sie als Whitebox-Methoden nur angewendet werden können, wenn die internen Zustände von gehosteten LLMs bereitgestellt oder Sprachmodelle lokal gehostet werden.

Des Weiteren kann die Bewertung korrekter und inkorrektur Aktionen uneindeutig sein. Wann Aktionen als korrekt gelten, wurde zwar zuvor im Ground Truth definiert, jedoch traten bei komplexen oder mehrdeutigen Interaktionen auch Fälle auf, bei denen eine Aktion nicht eindeutig als inkorrekt zu bewerten war. Denn auch wenn eine Referenzlösung erwartet wurde, gibt es durchaus verschiedene Pfade innerhalb der Website, um eine bestimmte Aufgabe zu absolvieren. Wurde beispielsweise erwartet, über das Navigationsmenü „Dienstleistungsformular“ ein neues Ticket anzulegen, das LLM jedoch in die Detailseite eines Kunden navigiert, so galt dieser Schritt als inkorrekt. In einer späteren Planung hat es jedoch trotzdem die Möglichkeit, wenn auch über Umwege, aus dem Kunden heraus für diesen ein Ticket zu erstellen.

Außerdem hängt die gemessene Unsicherheit nicht nur vom vorhandenen Wissen des LLMs ab, sondern auch von der Formulierung der Aufgabeninstruktionen oder der Gestaltung des Systemprompts. Der Systemprompt wurde immer wieder mit kleinen Änderungen verbessert, welche oft große Änderungen im Modellverhalten erzielten und die Konfidenzwerte beeinflussten. Dieses Verhalten wurde ebenfalls von Y. Huang et al., 2025 beobachtet. Auch das Hintergrundwissen selbst kann zu Verzerrungen führen. Insgesamt zeigte sich zwar eine bessere Trennschärfe der meisten Metriken, gleichzeitig kann es dadurch einen Bias erzeugen, wodurch sich das LLM sicherer ausdrückt, wie es bei der verbalisierten Konfidenz der Fall war.

Darüber hinaus beschränkte sich die Evaluation in dieser Arbeit auf die Trennschärfe der untersuchten Unsicherheitsmetriken durch den AUROC. Zukünftig könnte die Evaluation erweitert werden, indem man weitere Metriken heranzieht und zum Beispiel die Kalibrierung der Unsicherheitsmetriken untersucht. Dazu eignen sich insbesondere Metriken wie der Expected Calibration Error (ECE) oder Brier Score.

Abschließend ist zu erwähnen, dass alle Messungen und Untersuchungen unter Verwendung von GPT-4.1 von OpenAI durchgeführt wurden. Mit hoher Geschwindigkeit werden neue Sprachmodelle mit steigender Leistungsfähigkeit und Kapazität veröffentlicht. Vertraut man dem Trend, dass leistungsfähigere LLMs eine bessere Kalibrierung ihrer Vertrauensangaben

aufweisen, ist künftig mit einer steigenden Zuverlässigkeit der verbalisierten Metriken zu rechnen.

6.5. Praktische Implikationen

Die Ergebnisse dieser Arbeit zeigen deutlich, dass Unsicherheitsquantifizierung nicht nur als nachträgliche theoretische Analysen, sondern in realen, sicherheitskritischen Automatisierungen von Prozessen oder Systemen unmittelbar zur Steuerung eingesetzt werden kann. Vor allem konsistenzbasierte Verfahren eignen sich, um das Verhalten während automatisierter Aktionen zu lenken. Je nach Schwellenwert der resultierenden Unsicherheitsmetrik kann eine Aktion automatisch ausgeführt, Nutzer*innen vorgeschlagen oder vor der automatischen Ausführung zur Bestätigung vorgelegt werden. Doch nicht nur in Echtzeitszenarien sind die Verfahren sinnvoll einsetzbar. Sie können auch im Bereich Monitoring oder Qualitätssicherung nützlich sein, um Fehlerquellen oder Verbesserungen beziehungsweise Verschlechterungen eines Produktes oder einer Anwendung zu erkennen.

Auch tokenbasierte und selbst verbalisierte Verfahren stellen nützliche Unsicherheitsmethoden dar, die als alleinstehende oder unterstützende Signale zugrundeliegender Unsicherheit dienen können. Hintergrundwissen und klare Systeminstruktionen führen nachweislich zu einer besseren Trennschärfe der genannten Metriken. Dadurch ermöglichen die Methoden auch eine Abschätzung, ob KI-gestützte Systeme über genügend oder zu wenig Informationen über die zu erledigen Aufgabengebiete verfügen.

Aufgrund des hohen Ressourcenverbrauchs konsistenzbasierter Verfahren, erscheint es letztlich sinnvoll, Unsicherheitsmetriken als gestufte Verfahren zu implementieren. Bei kritischen Prozessen empfiehlt sich die konsistenzbasierte Unsicherheitsquantifizierung. Für weniger kritische Prozesse könnte man wiederum günstigere und ressourcenärmere Verfahren, wie verbalisierte Methoden oder semantische Konsistenz mit jedoch weniger erzeugten Samples, verwenden.

6.6. Ausblick

Im Hinblick auf weiterführende Forschung erscheint es sinnvoll, den Fokus künftiger Arbeiten verstärkt auf konsistenzbasierte Verfahren zur Unsicherheitsquantifizierung zu legen, da sich diese in der vorliegenden Untersuchung insgesamt als besonders leistungsfähig und zuverlässig erwiesen haben. Dabei sollte nicht allein ihre Leistungsfähigkeit überprüft werden. Ebenso relevant ist es, zugrundeliegende Schwächen weiter zu analysieren und letztlich zu reduzieren. Ein vielversprechender Ansatz findet sich bei Kuhn et al., [2022](#). Sie beschreiben

beispielsweise das Problem der semantischen Äquivalenz, wodurch Sprachmodelle völlig verschiedene Wörter nutzen können, welche inhaltlich jedoch dasselbe bedeuten. Dies führt zu hoher Entropie, welche letztlich fälschlicherweise als hohe Unsicherheit gedeutet wird. Sie erzeugen mehrere Samples und gruppieren diese nach ihrer Bedeutung. Die Entropie wird daraufhin nicht auf einzelnen Tokens, sondern auf zusammengefassten Bedeutungsgruppen berechnet. Dieser Ansatz stellt ein vielversprechendes Verfahren in Aussicht, welches die Konsistenz und Entropie als Ensemble zur Unsicherheitserkennung darstellt.

Ein weiterer Ansatz wird von M. Zhang et al., 2024 vorgestellt. Sie verfolgen ebenfalls einen konsistenzbasierten Ansatz, indem sie Konfidenz als Verkettung von Unsicherheit in Bezug auf die Frage und Treue bezüglich der generierten Antwort eines LLMs untersuchen. Um die Unsicherheit des LLMs zu erfassen, werden mehrere Antwortsamples zur gleichen Frage erzeugt und daraus eine Häufigkeitsverteilung ermittelt. Zusätzlich wird die Treue untersucht, indem das LLM eine Antwort aus mehreren vorgegebenen Optionen auswählt. Daraufhin wird der Prompt erneut abgeschickt und die Antwort wird ersetzt durch „All other options are wrong“. Wählt das LLM weiterhin diese Option, spricht das für eine hohe Treue. Für die in dieser Arbeit untersuchte Domäne ist dieser Gedanke interessant, da die verfügbaren UI-Elemente im Kern ebenfalls Handlungsoptionen darstellen, welche durch Aktionen ausgewählt werden. Es könnte geprüft werden, ob sich dieses Verfahren zur Bewertung der Unsicherheit bei generierten Aktionen ebenfalls anwenden lässt.

Die Ergebnisse legen zudem nahe, dass keine einzelne Metrik in allen untersuchten Bedingungen und Szenarien konsistent dominiert. Die zugrundeliegende Unsicherheit des Modells scheint durch die verschiedenen Metriken unterschiedlich abgebildet zu werden. Dementsprechend könnte man tiefergehend untersuchen, welche konkreten Einflussfaktoren die resultierenden Konfidenz- beziehungsweise Unsicherheitswerte maßgeblich beeinflussen.

Ein weiterer Punkt betrifft die Übertragbarkeit der angewendeten Metriken auf andere Sprachmodelle. Die Untersuchungen dieser Arbeit wurden ausschließlich an OpenAIs GPT-4.1 durchgeführt, das interne Logprobs bereitstellt. Um auch Whitebox- und Greybox-Methoden auf geschlossene Modelle anzuwenden, könnten zukünftig auch sogenannte „Surrogatmodelle“ (Stellvertreter) verwendet werden (Shrivastava et al., 2023). Dabei werden ein Blackbox- und ein Whiteboxmodell mit demselben Prompt gespeist. Die Tokenlogprobs des Whiteboxmodells werden dann stellvertretend verwendet, um die Konfidenz des Blackboxmodells zu bewerten.

Die Unsicherheitsquantifizierung in dieser Arbeit wurde innerhalb einer spezifischen Domäne mit eigenen Ground Truth Referenzlösungen untersucht. In einem nächsten Schritt würde es sich anbieten, diesen Anwendungskontext zu verlassen und die Unsicherheitsmetriken anhand allgemeiner Web-Agent-Benchmarks zu analysieren. Inzwischen haben sich hierfür Benchmarks, wie „Mind2Web“ (Deng et al., 2023), „Online-Mind2Web“ (Gou et al., 2025) oder

„WebArena“ (Zhou et al., 2024) etabliert, die Webagents über unterschiedliche Webseiten und Aufgaben bewerten.

Schließlich ist auch der Stichprobenumfang zu berücksichtigen. Zwar wurden mehrere Szenarien unter verschiedenen Bedingungen untersucht, jedoch bleibt die Anzahl der gesamten Aktionen innerhalb einer speziellen Anwendungsdomäne überschaubar. Weitere Untersuchungen sollten einerseits einen größeren Stichprobenumfang aufweisen, andererseits sollten die Untersuchungen in einer weiterreichenden Domäne durchgeführt werden. Auf diese Weise könnte untersucht werden, inwiefern sich die hier gefundenen Erkenntnisse auch auf andere themennahe Domänen anwenden und generalisieren lassen.

7. Fazit

Das Ziel dieser Arbeit war es, zu untersuchen, wie sich die Unsicherheit von Modellantworten in der Praxis erkennen lässt, sodass IT-Systeme programmatisch auf unsichere oder potenziell fehlerhafte Modellantworten reagieren können. Dazu wurden gängige Unsicherheitsmetriken aus drei unterschiedlichen Kategorien anhand verschiedener Aufgabenstellungen analysiert und verglichen. Um dies untersuchen zu können, wurde eine eigene Anwendung entwickelt, die es einerseits ermöglicht, eine Webseite im Browser automatisiert zu steuern und andererseits Messdaten zur Bewertung der zugrundeliegenden Unsicherheit erfasst.

Über alle Aufgaben hinweg erwiesen sich dabei vor allem konsistenzbasierte Unsicherheitsmetriken als zuverlässige Indikatoren für unsichere oder fehlerhafte Modellantworten. Für die verbalisierten Metriken konnte hingegen nicht bestätigt werden, dass sie den tokenbasierten Unsicherheitsmetriken durchgehend überlegen sind. Dennoch zeigte sich, dass alle untersuchten Metriken grundsätzlich als nützliche Unsicherheitsindikatoren dienen können.

Die Arbeit hat gezeigt, dass sich Verfahren der Unsicherheitsquantifizierung auch in einem praxisnahen Anwendungskontext differenziert untersuchen und vergleichen lassen. Die gewonnenen Erkenntnisse liefern damit vielversprechende Hinweise darauf, wie Unsicherheit in KI-gestützten Produktivsystemen künftig besser erkannt und berücksichtigt werden kann und sind zugleich richtungsweisend für weiterführende vertiefende Forschungsansätze. Damit leistet die Arbeit einen Beitrag zur sicheren Nutzung von LLMs in produktiven Systemen und zeigt zugleich, dass die zuverlässige Erkennung von Unsicherheit ein nicht abschließend gelöstes Problem bleibt.

Literatur

- Alammar, J. (2018, Juni). The Illustrated Transformer. Verfügbar 5. Januar 2026 unter <https://jalammar.github.io/illustrated-transformer/>
- Bahdanau, D., Cho, K., & Bengio, Y. (2016, Mai). Neural Machine Translation by Jointly Learning to Align and Translate [arXiv:1409.0473 [cs]]. <https://doi.org/10.48550/arXiv.1409.0473>
- Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., ... Amodei, D. (2020, Juli). Language Models are Few-Shot Learners [arXiv:2005.14165 [cs]]. <https://doi.org/10.48550/arXiv.2005.14165>
- Browser Use. (2026, April). Browser Use - The way AI uses the internet. Verfügbar 30. April 2026 unter <https://browser-use.com>
- Chen, J., & Mueller, J. (2024). Quantifying Uncertainty in Answers from any Language Model and Enhancing their Trustworthiness. In L.-W. Ku, A. Martins & V. Srikumar (Hrsg.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (S. 5186–5200). Association for Computational Linguistics. <https://doi.org/10.18653/v1/2024.acl-long.283>
- Deng, X., Gu, Y., Zheng, B., Chen, S., Stevens, S., Wang, B., Sun, H., & Su, Y. (2023). Mind2Web: Towards a Generalist Agent for the Web. *Advances in Neural Information Processing Systems*, 36, 28091–28114. Verfügbar 29. April 2026 unter https://proceedings.neurips.cc/paper_files/paper/2023/hash/5950bf290a1570ea401bf98882128160-Abstract-Datasets_and_Benchmarks.html
- Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019, Mai). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding [arXiv:1810.04805 [cs]]. <https://doi.org/10.48550/arXiv.1810.04805>
- Elsworth, C., Huang, K., Patterson, D., Schneider, I., Sedivy, R., Goodman, S., Townsend, B., Ranganathan, P., Dean, J., Vahdat, A., Gomes, B., & Manyika, J. (2025, August). Measuring the environmental impact of delivering AI at Google Scale [arXiv:2508.15734 [cs]]. <https://doi.org/10.48550/arXiv.2508.15734>

- Fadeeva, E., Rubashevskii, A., Shelmanov, A., Petrakov, S., Li, H., Mubarak, H., Tsymbalov, E., Kuzmin, G., Panchenko, A., Baldwin, T., Nakov, P., & Panov, M. (2024). Fact-Checking the Output of Large Language Models via Token-Level Uncertainty Quantification. In L.-W. Ku, A. Martins & V. Srikumar (Hrsg.), *Findings of the Association for Computational Linguistics: ACL 2024* (S. 9367–9385). Association for Computational Linguistics. <https://doi.org/10.18653/v1/2024.findings-acl.558>
- Geng, J., Cai, F., Wang, Y., Koepl, H., Nakov, P., & Gurevych, I. (2024, März). A Survey of Confidence Estimation and Calibration in Large Language Models [arXiv:2311.08298 [cs]]. <https://doi.org/10.48550/arXiv.2311.08298>
- Gou, B., Huang, Z., Ning, Y., Gu, Y., Lin, M., Qi, W., Kopanov, A., Yu, B., Gutiérrez, B. J., Shu, Y., Song, C. H., Wu, J., Chen, S., Moussa, H. N., Zhang, T., Xie, J., Li, Y., Xue, T., Liao, Z., ... Su, Y. (2025, Juli). Mind2Web 2: Evaluating Agentic Search with Agent-as-a-Judge [arXiv:2506.21506 [cs]]. <https://doi.org/10.48550/arXiv.2506.21506>
- Huang, H.-Y., Yang, Y., Zhang, Z., Lee, S., & Wu, Y. (2024, Oktober). A Survey of Uncertainty Estimation in LLMs: Theory Meets Practice [arXiv:2410.15326 [cs]]. <https://doi.org/10.48550/arXiv.2410.15326>
- Huang, Y., Song, J., Wang, Z., Zhao, S., Chen, H., Juefei-Xu, F., & Ma, L. (2025). Look Before You Leap: An Exploratory Study of Uncertainty Measurement for Large Language Models [arXiv:2307.10236 [cs]]. *IEEE Transactions on Software Engineering*, 51(2), 413–429. <https://doi.org/10.1109/TSE.2024.3519464>
- IBM. (2021, Oktober). Was ist ein neuronales Netz? | IBM. Verfügbar 5. Dezember 2025 unter <https://www.ibm.com/de-de/think/topics/neural-networks>
- Kadavath, S., Conerly, T., Askell, A., Henighan, T., Drain, D., Perez, E., Schiefer, N., Hatfield-Dodds, Z., DasSarma, N., Tran-Johnson, E., Johnston, S., El-Showk, S., Jones, A., Elhage, N., Hume, T., Chen, A., Bai, Y., Bowman, S., Fort, S., ... Kaplan, J. (2022, November). Language Models (Mostly) Know What They Know [arXiv:2207.05221 [cs]]. <https://doi.org/10.48550/arXiv.2207.05221>
- Kipp, M. (2025, März). Neuronale Netze und Deep Learning. Verfügbar 2. Januar 2026 unter <https://www.michaelkipp.de/deeplearning/>
- Kiureghian, A. D., & Ditlevsen, O. (2009). Aleatory or epistemic? Does it matter? *Structural Safety*, 31(2), 105–112. <https://doi.org/10.1016/j.strusafe.2008.06.020>
- Kuhn, L., Gal, Y., & Farquhar, S. (2022). Semantic Uncertainty: Linguistic Invariances for Uncertainty Estimation in Natural Language Generation. Verfügbar 4. August 2025 unter <https://openreview.net/forum?id=VD-AYtP0dve>
- Lin, S., Hilton, J., & Evans, O. (2022). Teaching Models to Express Their Uncertainty in Words. *Transactions on Machine Learning Research*. Verfügbar 4. August 2025 unter <https://openreview.net/forum?id=8s8K2UZGTZ>
- Ma, H., Chen, J., Zhou, J. T., Wang, G., & Zhang, C. (2025, Mai). Estimating LLM Uncertainty with Evidence [arXiv:2502.00290 [cs]]. <https://doi.org/10.48550/arXiv.2502.00290>

- Manakul, P., Liusie, A., & Gales, M. J. F. (2023, Oktober). SelfCheckGPT: Zero-Resource Black-Box Hallucination Detection for Generative Large Language Models [arXiv:2303.08896 [cs]]. <https://doi.org/10.48550/arXiv.2303.08896>
- Microsoft. (2026a, Februar). Fast and reliable end-to-end testing for modern web apps | Playwright Python. Verfügbar 16. Februar 2026 unter <https://playwright.dev/python/>
- Microsoft. (2026b, Februar). Visual Studio Code. Verfügbar 26. Februar 2026 unter <https://code.visualstudio.com/>
- Mikolov, T., Yih, W.-t., & Zweig, G. (2013). Linguistic Regularities in Continuous Space Word Representations. In L. Vanderwende, H. Daumé III & K. Kirchhoff (Hrsg.), *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (S. 746–751). Association for Computational Linguistics. Verfügbar 2. Januar 2026 unter <https://aclanthology.org/N13-1090/>
- Minaee, S., Mikolov, T., Nikzad, N., Chenaghlu, M., Socher, R., Amatriain, X., & Gao, J. (2025, März). Large Language Models: A Survey [arXiv:2402.06196 [cs]]. <https://doi.org/10.48550/arXiv.2402.06196>
- Mora-Cross, M., & Calderon-Ramirez, S. (2024). Uncertainty Estimation in Large Language Models to Support Biodiversity Conservation. In Y. Yang, A. Davani, A. Sil & A. Kumar (Hrsg.), *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 6: Industry Track)* (S. 368–378). Association for Computational Linguistics. <https://doi.org/10.18653/v1/2024.naacl-industry.31>
- OpenAI. (2025a, April). GPT-4.1 Model | OpenAI API. Verfügbar 26. Februar 2026 unter <https://developers.openai.com/api/docs/models/gpt-4.1>
- OpenAI. (2025b, April). Introducing GPT-4.1 in the API. Verfügbar 5. Februar 2026 unter <https://openai.com/index/gpt-4-1/>
- Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language Models are Unsupervised Multitask Learners. Verfügbar 4. August 2025 unter <https://www.semanticscholar.org/paper/Language-Models-are-Unsupervised-Multitask-Learners-Radford-Wu/9405cc0d6169988371b2755e573cc28650d14dfe>
- Raschka, S. (2025). *Build a Large Language Model (from Scratch)*. Manning Publications Co. LLC.
- Reimers, N., & Gurevych, I. (2019, August). Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks [arXiv:1908.10084 [cs]]. <https://doi.org/10.48550/arXiv.1908.10084>
- Sahoo, P., Singh, A. K., Saha, S., Jain, V., Mondal, S., & Chadha, A. (2025, März). A Systematic Survey of Prompt Engineering in Large Language Models: Techniques and Applications [arXiv:2402.07927 [cs]]. <https://doi.org/10.48550/arXiv.2402.07927>
- Schieb, J. (2025, Oktober). Studie: KI-Chatbots machen bei 40 Prozent der Antworten Fehler. Verfügbar 3. Februar 2026 unter <https://www.tagesschau.de/wissen/technologie/kuenstliche-intelligenz-fakten-100.html>

- Schlüter, M. (2016). Neuronale Netze. <https://doi.org/10.13140/RG.2.2.16261.55523>
- Shorinwa, O., Mei, Z., Lidard, J., Ren, A. Z., & Majumdar, A. (2025, Juli). A Survey on Uncertainty Quantification of Large Language Models: Taxonomy, Open Research Challenges, and Future Directions [arXiv:2412.05563 [cs]]. <https://doi.org/10.48550/arXiv.2412.05563>
- Shrivastava, V., Liang, P., & Kumar, A. (2023, November). Llamas Know What GPTs Don't Show: Surrogate Models for Confidence Estimation [arXiv:2311.08877 [cs]]. <https://doi.org/10.48550/arXiv.2311.08877>
- Tian, K., Mitchell, E., Zhou, A., Sharma, A., Rafailov, R., Yao, H., Finn, C., & Manning, C. (2023). Just Ask for Calibration: Strategies for Eliciting Calibrated Confidence Scores from Language Models Fine-Tuned with Human Feedback. In H. Bouamor, J. Pino & K. Bali (Hrsg.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing* (S. 5433–5442). Association for Computational Linguistics. <https://doi.org/10.18653/v1/2023.emnlp-main.330>
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017, Juni). Attention Is All You Need [arXiv:1706.03762 [cs]]. <https://doi.org/10.48550/arXiv.1706.03762>
- Vijayakumar, A. K., Cogswell, M., Selvaraju, R. R., Sun, Q., Lee, S., Crandall, D., & Batra, D. (2018, Oktober). Diverse Beam Search: Decoding Diverse Solutions from Neural Sequence Models [arXiv:1610.02424 [cs]]. <https://doi.org/10.48550/arXiv.1610.02424>
- Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., Narang, S., Chowdhery, A., & Zhou, D. (2023, März). Self-Consistency Improves Chain of Thought Reasoning in Language Models [arXiv:2203.11171 [cs]]. <https://doi.org/10.48550/arXiv.2203.11171>
- Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., & Zhou, D. (2023, Januar). Chain-of-Thought Prompting Elicits Reasoning in Large Language Models [arXiv:2201.11903 [cs]]. <https://doi.org/10.48550/arXiv.2201.11903>
- Xiong, M., Hu, Z., Lu, X., Li, Y., Fu, J., He, J., & Hooi, B. (2024, März). Can LLMs Express Their Uncertainty? An Empirical Evaluation of Confidence Elicitation in LLMs [arXiv:2306.13063 [cs]]. <https://doi.org/10.48550/arXiv.2306.13063>
- Yang, J., Wang, Z., Lin, Y., & Zhao, Z. (2024, August). Large Language Model Tokenizer Bias: A Case Study and Solution on GPT-4o [arXiv:2406.11214 [cs] version: 2]. <https://doi.org/10.48550/arXiv.2406.11214>
- Yang, Y., Yoo, H., & Lee, H. (2025). MAQA: Evaluating Uncertainty Quantification in LLMs Regarding Data Uncertainty. In L. Chiruzzo, A. Ritter & L. Wang (Hrsg.), *Findings of the Association for Computational Linguistics: NAACL 2025* (S. 5846–5863). Association for Computational Linguistics. <https://doi.org/10.18653/v1/2025.findings-naacl.325>
- Zhang, M., Huang, M., Shi, R., Guo, L., Peng, C., Yan, P., Zhou, Y., & Qiu, X. (2024, Oktober). Calibrating the Confidence of Large Language Models by Eliciting Fidelity [arXiv:2404.02655 [cs]]. <https://doi.org/10.48550/arXiv.2404.02655>

- Zhang, T., Kishore, V., Wu, F., Weinberger, K. Q., & Artzi, Y. (2020, Februar). BERTScore: Evaluating Text Generation with BERT [arXiv:1904.09675 [cs]]. <https://doi.org/10.48550/arXiv.1904.09675>
- Zhou, S., Xu, F. F., Zhu, H., Zhou, X., Lo, R., Sridhar, A., Cheng, X., Ou, T., Bisk, Y., Fried, D., Alon, U., & Neubig, G. (2024, April). WebArena: A Realistic Web Environment for Building Autonomous Agents [arXiv:2307.13854 [cs]]. <https://doi.org/10.48550/arXiv.2307.13854>

A. Anhang

A.1. Automatisierungsanwendung

Codeblock A.1: Fortschrittsabhängige Systemprompts

```
1 {
2   _base_prompt = """
3   - You are an AI agent that automates browser tasks by planning a sequence of
4     actions to control web applications.
5   - You receive a list of interactive elements along with their XPath information.
6   - You can click all elements listed, except for 'text-content'-elements. Also use
7     Divs and Spans.
8   - Use multi-step reasoning to fulfill the user request.
9   - Always respond in JSON format with a list of steps.
10  - Use the 'xpath' value only if you are absolutely sure. Otherwise, set '
11    need_replanning' = True.
12  - If UI information is missing or unclear (e.g., after a page change), still plan
13    the action in the sequence and set 'need_replanning' = True
14  - If you dont find a matching element for a provided information you extracted
15    from the user input, ignore it.
16  - Each step must include:
17    - 'description': a short description of the action,
18    - 'confidence': a value between 0 and 100 indicating your confidence that
19      this action is correct and leads to solving the ultimate task,
20    - 'confidence_in_words': describe your confidence that this action is correct
21      and leads to solving the ultimate task with one of these 5 words 'lowest', 'low
22      ', 'medium', 'high', 'highest',
23  - Keep your response as short as possible.
24  - If the task is complete and no further actions need to be performed, set '
25    all_actions_performed' = True and finish the sequence with an ActionResult where
26    you set only the description, confidence and 'is_terminal' = True.
27  - Only click a tab or button to expand a collapsable element if its content is not
28    already visible.
29  - Never ask for the visible input fields or Xpath of an element.
```

```
19 - If you detect that you already tried an approach multiple times, try a different
    approach with the actions.
20 """
21
22 if not appsettings.USE_LOGPROBS:
23     _base_prompt += "- Always respond using tools or function calls.\n"
24
25 initial_prompt = _base_prompt + """
26 - Analyze the list of interactive elements (with XPath), the given examples (
    routines), and the user input to generate an appropriate action sequence.
27 - Pay particular attention to missing or incomplete UI information. Only plan as
    far as is safely possible.
28 """
29
30 replan_prompt = _base_prompt + """
31 - You re-analyze the list of interactive elements, already performed actions,
    routines and the current user input to extend or correct a previously started
    but not finished executable sequence.
32 - The previous plan is included in the assistant role message.
33 - Steps with 'executed': true have already been carried out
34 - Only steps with 'executed': false may be modified or added.
35 - If new information requires additional steps, append them to the end.
36 - If a page change occurred or UI elements no longer match, set 'need_replanning'
    = True.
37 """
38
39 final_assessment_prompt = replan_prompt + """
40 - All planning steps have been performed. Assess whether everything has been
    completed or whether further steps are necessary.
41 - If this is the case, add further steps.
42 """
43 }
```

A.2. Hintergrundwissen

1. Wenn es sich bei der Seite um ein Formular handelt, extrahiere alle Informationen aus der Nutzereingabe und versuche sie den vorhandenen Eingabefeldern zuzuweisen.
2. Wenn du einen Akteur auswählen willst, kannst du seine Nummer direkt in das entsprechende Feld eingeben oder über den Button rechts neben dem Nummernfeld ein erweitertes Suchfenster öffnen.
3. Das Dienstleistungsformular dient der Auftragserfassung. Nach einem Klick auf das Menü muss entweder 'Serviceanfrage' oder 'Angebotsanfrage' ausgewählt werden. Handelt es sich nicht um eine Angebotsanfrage, wähle 'Serviceanfrage'. Jetzt erst öffnet sich ein Formular mit Eingabefeldern. Wenn es sich um eine Serviceanfrage handelt, der Benutzer aber keinen Serviceeinsatz angegeben hat, muss als Fallback immer Serviceeinsatz = Reparatur ausgewählt werden.
4. Wenn du aufgefordert wirst, einen Akteur, wie Kunde oder Mitarbeiter, aufzurufen, nutze das Menü Adressbuch. Hier findest du eine Auswahl von Eingabefeldern. Wähle dazu die Art des Akteurs aus, indem du 'Kunde' oder 'Mitarbeiter' auswählst. Drücke danach auf den Suchen-Button.
5. Ein neues Angebot kann über das Menü Angebote erstellt werden. Wenn während der Erstellung oder Bearbeitung eines Angebots oder Kostenvoranschlags Artikel hinzugefügt werden sollen, so gibt es dafür zwei Möglichkeiten. Wenn die Artikelnummer vorliegt, kann diese direkt in das Eingabefeld 'Artikelnummer' eingegeben und mit Klick auf das 'Plus' hinzugefügt werden. Alternativ kann das Artikelsuchfenster durch Klick 'Artikelsuche öffnen' geöffnet werden. Dort gibt es verschiedene Filteroptionen für eine gezieltere Artikelsuche.
6. Über das Menü Inventare gelangst du zu einer Suchmaske für Inventare. Nach der Suche erschienen Inventarergebnisse. Drücke auf das Editfeld oder die ID, um zu der Detailseite zu navigieren.
7. Artikeldetails können am einfachsten in der Artikelverwaltung unter dem Administrationsmenü eingesehen werden. Alternativ kann dies innerhalb eines Angebots geschehen.
8. In das Menü 'Organisationsstruktur' gelangst du über das Menü 'Administratoren', dann 'Systemeinstellungen'.
9. Bei dem Anlegen neuer Qualifikationen füge zuerst durch den Button 'Neuer Eintrag' einen leeren Eintrag hinzu und bearbeite ihn danach entsprechend.

A.3. Ground Truth Labels der Interaktionsszenarien

S1: Dienstleistungsformular

1. Click: Dienstleistungsformular
2. Click: Serviceanfrage
3. Click + Input: Kundennummer
4. Click + Input: Ansprechpartner
5. Click + Input: Problembeschreibung
6. Click + Select: Serviceeinsatz für 'Reparatur'
7. Click: Erfassen und versenden

S2: Adressbuch

1. Click: Adressbuch
2. Click + Select: Typ 'Kunde'
3. Click + Input: Firma 'Absolute'
4. Click: Icon 'ABSOLUTE SOFTWARE GMBH'

S3: Angebot

1. Click: Angebote
2. Click: Neues Angebot
3. Click + Input: Kundennummer
4. Click + Input: Abweichende Kontakt E-Mail
5. Click + Input: Ansprechpartner
6. Click + Input: Referenz (Kunde)
7. Click + Input: Titel
8. Click + Input: Betreff
9. Click: Artikelsuche öffnen
10. Click + Input: Bezeichnung
11. Click: Suche mit Auswahl
12. Click: Artikel 'KOLBENKOMPRESSOR AIRPROFI'
13. Click + Input: Erfassen

S4: Inventar

1. Click: Inventar
2. Click + Input: Kundennummer (optional)

3. Click + Input: Seriennummer
4. Click: Auswahl Inventar AVL MDS 105 ZYL56182
5. Click: Wartungsvertrag / Wartungslauf besteht 'Ja'
6. Click + Select: Vertrag / Plan 'nach Vertrag'
7. Click + Select: Wartungsintervall 'jährlich'
8. Click + Select: ab Monat 'August'
9. Click + Select: ab Jahr '2025'
10. Click: Speichern

S5: Artikel

1. Click: Administratoren
2. Click: Systemeinstellungen
3. Click: Artikelverwaltung
4. Click + Input: Bezeichnung
5. Click: Suche mit Auswahl
6. Terminal: Angabe von Verkaufspreis für Geodyna 7750P Radauswuchtmaschine

S6: Organisationsstruktur

1. Click: Administratoren
2. Click: Systemeinstellungen
3. Click: Organisationsstruktur
4. Click + Input: Name 'TSC-302'
5. Click: Suche mit Auswahl
6. Click: Edit TSC-302 Ottobrunn
7. Click: Actorauswahl von 'Verantwortlich für die Disposition'
8. Click + Input: Vorname 'Peter'
9. Click + Input: Nachname: 'Mustermann'
10. Click: Suche mit Auswahl
11. Click: Auswahl von 558589 'Peter Mustermann'
12. Speichern

S7: Qualifikationen

1. Click: Adressbuch
2. Click + Select: Typ 'Mitarbeiter' (optional)
3. Click + Input: Vorname 'Peter'
4. Click + Input: Nachname 'Mustermann'

5. Click: 558589 'Peter Mustermann'
6. Click: Qualifikationen
7. Click: Delete 'AHS Bremsenprüfstände Grundschulung'
8. Click: Hinzufügen
9. Click + Select: Zeile 1 Qualifikation 'Klimasachkundennachweis'
10. Click + Select: Zeile 1 Level 'Theorie'
11. Click: Delete 'AHS Bremsenprüfstände Grundschulung'
12. Click: Speichern

A.4. Deskriptive Statistiken der zusammengefassten Ergebnisse

Tabelle A.1.: Deskriptive Statistik der erfassten, zusammengefassten Konfidenzmesswerte. Die Tabelle zeigt die Anzahl aller Messwerte (count), den durchschnittlichen Konfidenzwert (mean), die Standardabweichung (std), den niedrigsten (min) und höchsten (max) Messwert, sowie die drei verschiedenen Perzentile (25 %, 50 %, 75 %). Insgesamt wurden 5409 Aktionen durchgeführt und erfasst.

	verbal	consistency	max (H)	avg (H)	min (p)	avg (p)
count	5409.00	5409.00	5409.00	5409.00	5409.00	5409.00
mean	88.21	78.83	83.06	95.58	73.26	93.58
std	13.12	14.01	24.52	6.72	38.57	11.79
min	0.00	27.68	4.83	48.13	0.00	0.38
25%	85.00	70.68	64.00	91.97	33.55	89.61
50%	92.00	81.67	99.92	99.93	99.97	99.98
75%	95.00	89.64	100.00	100.00	100.00	100.00
max	100.00	100.00	100.00	100.00	100.00	100.00

Eigenständigkeitserklärung

Während der Erstellung dieser Masterarbeit wurde ChatGPT von OpenAI als Hilfsmittel zur sprachlichen Verbesserung und zur Unterstützung einzelner Formulierungen eingesetzt. Die Nutzung erfolgte nicht zur eigenständigen Erstellung wissenschaftlicher Inhalte. Alle KI-gestützten Vorschläge wurden von mir geprüft, überarbeitet und nur übernommen, soweit sie meiner eigenen fachlichen Einschätzung entsprachen. Die Verantwortung für den Inhalt liegt vollständig bei mir.

Hiermit versichere ich, dass ich die vorliegende Masterarbeit mit dem Titel

Unsicherheitsquantifizierung von Large Language Models bei der Automatisierung und Steuerung interaktiver Webprozesse

selbstständig und nur mit den angegebenen Hilfsmitteln verfasst habe. Alle Passagen, die ich wörtlich aus der Literatur oder aus anderen Quellen wie z. B. Internetseiten übernommen habe, habe ich deutlich als Zitat mit Angabe der Quelle kenntlich gemacht.

Ort

Datum

Unterschrift im Original