

Freie Universität Berlin

Bachelorarbeit am Institut für Informatik der Freien Universität Berlin

Arbeitsgruppe Software Engineering

Inwiefern ist ChatGPT in der Lage, domänenspezifischen Code zu generieren?

John F. Jung

Matrikelnummer: 5332786

johnj67@zedat.fu-berlin.de

Betreuer/in: Prof. Dr. Lutz Prechelt

Eingereicht bei: Prof. Dr. Lutz Prechelt

Zweitgutachter/in: Prof. Dr. Claudia Müller-Birn

Berlin, 9. Oktober 2023

Zusammenfassung

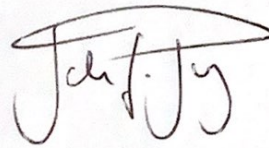
Die Fähigkeit von ChatGPT Code generieren zu können, sorgte im Bereich der Softwareentwicklung für ein enormes Interesse. Bekannte oder triviale Probleme kann ChatGPT mühelos lösen. Probleme in der Softwareentwicklung sind meist jedoch komplexer und vor allem einzigartig. Gemeint ist damit, dass teilweise Code für spezifische Domänen produziert werden muss, die sehr bestimmte Anforderungen besitzen. In dieser Arbeit gilt es herauszufinden, inwiefern es möglich ist, mit ChatGPT domänenspezifischen Code zu generieren. Die Aufgabe ist es, Flugregeln und Gesetze in Python Funktionen umzuwandeln, damit diese in die eigene Codebase der Lufthansa Systems integriert werden können. Eine große Rolle spielt dabei das Prompt Engineering. Durch Aneignung verschiedener Techniken und Methoden konnte so mithilfe von gezielten Prompts domänenspezifischer Code erstellt werden. Eine anschließende Studie ergab dabei, dass die individuellen Erfahrungen der Teilnehmer eine Rolle bei der Generierung von korrektem Code spielten.

Eidesstattliche Erklärung

Ich versichere hiermit an Eides Statt, dass diese Arbeit von niemand anderem als meiner Person verfasst worden ist. Alle verwendeten Hilfsmittel wie Berichte, Bücher, Internetseiten oder ähnliches sind im Literaturverzeichnis angegeben, Zitate aus fremden Arbeiten sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

John F. Jung

Datum: 9. Oktober 2023

A handwritten signature in black ink, appearing to read 'John F. Jung', with a large, sweeping horizontal stroke above the letters.

Inhaltsverzeichnis

1	Einführung	7
2	Grundlagen	8
2.1	Künstliche Intelligenz	8
2.2	ChatGPT	9
2.3	Warum ChatGPT	11
2.4	Netline/Crew Optimierer	11
2.5	Flugregeln und Gesetze	12
3	Verwandte Arbeiten	12
4	Prompt Engineering	13
4.1	Schlüsselemente eines Prompts	14
4.2	Prompt Techniken	14
5	Domänenspezifisches Prompting	17
5.1	ChatGPT Input	17
5.2	Nutzung der Prompts	21
5.3	Validierung des Skripts	23
6	Ergebnisse	24
6.1	Studienergebnisse	25
6.2	Skript für Lufthansa Systems	30
7	Fazit	30
A	Anhang	34
A.1	Erste Version des Skripts	34
A.2	Letzte Version des Skripts	38
A.3	Regeldokument	42
A.4	Skript für Lufthansa Systems	44

1 Einführung

Seit Anfang des Jahres 2023 hat eine neue Anwendung, vor allem im Bereich der Softwareentwicklung, in den verschiedensten Medien für große Aufmerksamkeit gesorgt. Gemeint ist ChatGPT¹ von OpenAI². Hinter ChatGPT verbirgt sich ein Chatbot, welcher auf Künstlicher Intelligenz basiert. Einer der Gründe für das enorme Interesse an dieser Anwendung ist der menschenähnliche Dialog mit einem Chatbot. Dieser kann auf diverse Fragen jeglicher Art eine Antwort geben und dem Nutzer bei verschiedensten Problemen helfen. So kann er innerhalb von Sekunden sogar mit Beachtung von Ernährungsarten einen Essensplan für die nächsten Tage erstellen oder einem dabei helfen, eine wichtige E-Mail an jemanden zu verfassen. Neben Essensplänen und E-Mails kann Chat GPT auch Code generieren. Nun kann man anstatt stundenlang über ein Software Problem nachzudenken, einfach ChatGPT sein Problem schildern und eine mögliche Lösung erhalten. Triviale Probleme und bekannte Fragen wie z.B. das Öffnen einer Datei oder das Löschen eines bestimmten Elements in einer Liste meistert der Chatbot einwandfrei. Oft werden jedoch Softwareentwickler mit Problemen und Fragen konfrontiert, die sich aufs eigene Softwareprojekt oder Produkt beziehen. Nicht selten kann so ein Projekt sehr groß und von einer eigenen Architektur geprägt sein, welche sich durch einzigartige Komponenten und Abhängigkeiten auszeichnen.

In dieser Bachelorarbeit möchte ich ermitteln, inwiefern ChatGPT in der Lage ist, auch über die simplen Probleme hinaus domänenspezifischen Code für eine bereits vorhandene Codebase zu generieren. Genauer sollen Flugregeln und Gesetze in Python von ChatGPT erzeugt werden, welche daraufhin in die Codebase übernommen werden können. Die Regeln sind dabei ein wichtiger Bestandteil des Netline/Crew Optimierer der Lufthansa Systems, da diese Auswirkungen auf den Optimierungsprozess der Crew Planung haben. Maximale Arbeitszeiten der Piloten oder Mindestruhezeiten für die Crew sind Beispiele für solche Regeln, die beim Optimieren beachtet werden müssen.

Aus den gesammelten Erkenntnissen werden sogenannte "base" Prompts in ein Skript eingebaut. Das Skript dient dabei als Anleitung für das Nutzen der resultierenden Prompts um damit domänenspezifischen Code zu erstellen. Mithilfe einer Studie wird das Skript zum einen auf die Nutzbarkeit validiert und zum anderen wird überprüft, ob es damit überhaupt möglich ist ohne weitere Hilfestellung Code zu generieren. Die Herausforderung wird es sein, dem Chatbot alle wichtigen Informationen zu übergeben, die es braucht, um korrekte Antworten zu erstellen.

In der Arbeit werde ich zuerst den Begriff Künstliche Intelligenz definieren und anschließend das Modell ChatGPT kurz erklären. Ebenfalls stelle ich den Optimierer der Lufthansa System vor, sowie die Flugregeln, die schließlich mit der Hilfe von ChatGPT generiert werden sollen. Daraufhin werden verwandte Arbeiten präsentiert, die sich ähnlich mit dem Thema beschäftigt haben. Im Kapitel Prompt Engineering werden die Schlüsselemente eines effektiven Prompts vorgestellt und bekannte Techniken. Mit den gesammelten Wissen werden im Hauptteil das Skript für die Studie selbst und vor allem der Kern, nämlich das Grundgerüst für die domänenspe-

¹<https://chat.openai.com>

²<https://openai.com/>

zifischen Prompts erläutert. Danach werden die Studienergebnisse ausgewertet und das letztendliche Skript für die Lufthansa Systems vorgestellt. Am Ende werde ich im Fazit die Forschungsfrage beantworten und das Potential von ChatGPT diskutieren.

2 Grundlagen

In diesem Kapitel werden die Begriffe Künstliche Intelligenz und ChatGPT genauer erläutert. Zudem werde ich meine Wahl bezüglich der Anwendung ChatGPT begründen und daraufhin das Produkt der Lufthansa Systems kurz vorstellen. Zum Schluss werden die Flugregeln und Gesetze mithilfe von Beispielen eingeführt.

2.1 Künstliche Intelligenz

Der Chatbot ChatGPT beruht wie bereits erwähnt auf Künstliche Intelligenz. Um den Aufbau und die Vorgehensweise von ChatGPT erläutern zu können, müssen vorab einige wichtige Begriffe geklärt werden. Einer dieser Begriffe ist Künstliche Intelligenz, kurz KI. KI umfasst die Kapazitäten einer Maschine, menschenähnliche Fähigkeiten wie logisches Denken, Lernvermögen, Planen und kreative Prozesse nachzuahmen[7]. Gleichzeitig dient KI als Oberbegriff für weitere Technologien wie Machine Learning[11], Natural Language Processing[3], Computer Vision[6] und Robotik[16]. Zudem gibt es zwei Arten von Künstlicher Intelligenz, die schwache bzw. schmale und starke Künstliche Intelligenz[15]. Die meisten KI Modelle sind nur schmale KI, darunter fallen auch vermeintlich komplexe Systeme wie ChatGPT selbst. Schmale KI zeichnet sich dadurch aus, auf bestimmte Aufgaben trainiert und ausgerichtet zu sein. Andere Anwendungen, die unter diese Art von KI fallen, sind z.B. Apples Siri, Amazons Alexa oder autonome Fahrzeuge. Starke KI hingegen beschreibt eine theoretische Form der KI, welche eine menschenähnliche Intelligenz erreichen könnte. Dazu hat es ein eigenes Bewusstsein, mit dem es in der Lage wäre, eigene komplexe Probleme zu lösen, neue Sachen zu lernen und zukünftige Ereignisse zu planen[2]. Ebenfalls ist zu erwähnen, dass der Chatbot auf die verschiedenen bereits angesprochenen KI Technologien zugreift. Eine Kombination aus Machine Learning, Deep Learning[8] und Natural Language Processing bilden die Grundlage für das Produkt, dabei ist Deep Learning ein Teilbereich von Machine Learning[13]. Beim Deep Learning kommen Algorithmen zum Einsatz, die auf diverse Arten von Analysetechniken und Lernmethoden angewendet werden. Es besteht aus einem neuronalen Netz, welches aus mindestens drei oder mehr Schichten besteht und in jeder Schicht sind Knoten miteinander verbunden[2]. Das neuronale Netz versucht dabei die Eigenschaften des Menschens zu imitieren und aus großen Datensätzen zu lernen[2]. Somit sind solche Modelle in der Lage, von sich aus zu lernen und die gewonnenen Erkenntnisse wieder mit neuen Inhalten zu verknüpfen. Dabei greift der Mensch nicht mehr beim Lernvorgang ein und die Maschine analysiert die rohen Daten und Informationen von alleine[5]. Die Verarbeitung natürlicher Sprache (Natural Language Processing, kurz NLP) ermöglicht Computern die Fähigkeit Texte und gesprochene Wörter zu verstehen. Darüber hinaus kann menschliche Sprache in eine für Computer verständliche Form übersetzt werden, damit darauf Algorithmen angewendet werden können[2]. So können Texte

generiert oder Gespräche in Form von Sprachassistenten wie Siri von Apple sogar geführt werden. Dies ist vor allem für einen Chatbot von hoher Bedeutung.

2.2 ChatGPT

ChatGPT steht für Chat Generative Pretrained Transformer und ist ein sogenanntes Large Language Model, kurz LLM[13]. LLM's sind KI Systeme die auf natürliche Sprache trainiert worden sind. Sie basieren auf neuronalen Netzwerken, die entworfen wurden, um Antworten auf diverse Sequenzen von gesprochenen Sprachen, Programmiersprachen und mathematischen Gleichungen zu beantworten[19]. Die Neuronen, welche die Knoten in der jeweiligen Schicht des Netzwerkes sind, sind die Stellen im System, in denen Informationen verarbeitet und weitergeleitet werden. So wird im neuronalen Netzwerk ein Text in der Form eines Strings als Input genommen und anschließend ein darauf beruhender Output generiert wie in Abbildung 1 visualisiert wird. Dies lässt den Anschein erwecken, dass solche Modelle unsere Sprache als solches verstehen und darauf antworten, so wie es ein Mensch tun würde. Jedoch wird der Input als aller erstes in numerische Daten codiert, damit dann anschließend im Netzwerk mit den Informationen gearbeitet werden kann.

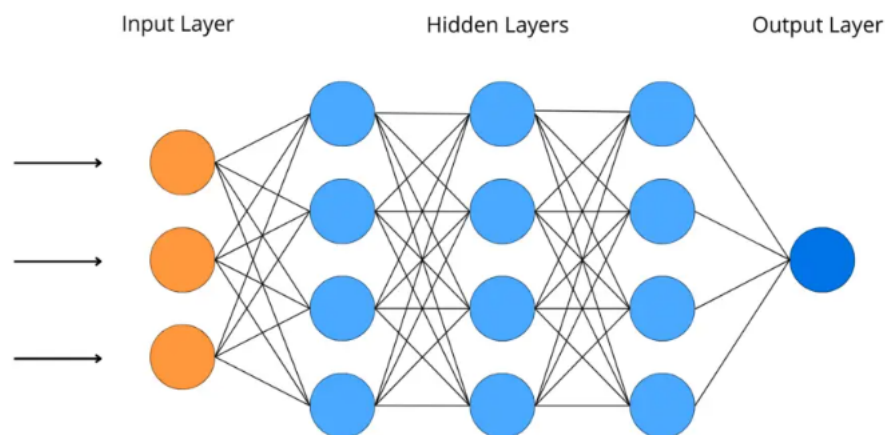


Abbildung 1: Neural Network Architecture
[14]

Die Vorgehensweise von ChatGPT's Output erfolgt sequentiell ab. So generiert es Wort für Wort seine Sätze, wobei das folgende Wort vom vorherigen Wort abhängt (Abbildung: 2). Es gibt dabei verschiedene Wahrscheinlichkeiten, mit denen ein Wort als Fortsetzung des Satzes genommen wird. Dies ist unter anderem der Grund dafür, dass bei gleicher Frage unterschiedliche Outputs generiert werden können.

Der Grundbaustein für das Berechnen dieser Wahrscheinlichkeiten ist die Transformer Architektur[19]. Innerhalb der Neuronen Schichten (Abb.: 1) werden mithilfe von mathematischen Algorithmen die jeweiligen Eventualitäten der Wörter berechnet. Die Technik, die in so einer Transformer Architektur eingesetzt wird, nennt sich "self-attention Mechanismus", womit es möglich ist, die Wichtigkeit der Wörter innerhalb einer Sequenz voneinander abzuwägen[20]. Des Weiteren kann durch die

2. Grundlagen

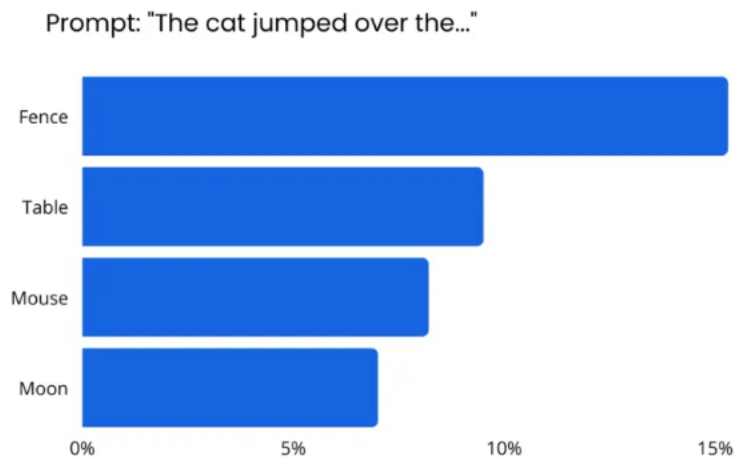


Abbildung 2: Wahrscheinlichkeit für Generierung eines bestimmten Wortes. Y-Achse: Wörter, X-Achse: Wahrscheinlichkeiten

[14]

Transformer Architektur, auf die großen, mit spezieller Kennzeichnung aufbereiteten Datensätze verzichtet werden, die eine Menge Zeit und Arbeit in der Vergangenheit beansprucht haben[20]. Die Transformer erkennen Muster und Strukturen mathematisch in den Daten selbst, wodurch es ermöglicht wird, viel größere Datensätze wie Bilder, Texte oder Informationen in verschiedenen Datenbanken zu benutzen. Diese Art des Trainings von Daten nennt sich auch unsupervised Learning[2], und stellt somit das Pretraining dar, so wie es im Namen steht.

Der eigentliche Trainingsprozess beginnt jetzt mit dem sogenannten "fine tuning" der Daten, da das Modell bisher nur darauf trainiert war, das nächste Wort vorherzusehen[13]. Um darüber hinaus sinnvolle, personalisierte, genaue und vor allem sichere Antworten auf die Fragen eines Users zu geben, benötigt es Verfeinerung. OpenAI, die Ersteller von ChatGPT, definieren dabei drei wesentliche Schritte[13](Abbildung: 3). Im ersten Schritt wird das Modell mit dem "supervised learning" trainiert. Input und Output werden von Menschen erstellt. So lernt das Modell diese Art von menschlichen Antworten kennen, um seine eigenen Antworten aus dem Pretraining daran anzupassen und zu überarbeiten. Daraufhin werden im zweiten Schritt die neuen Antworten von Menschen bewertet und je nach Genauigkeit und Sicherheit eingeordnet. Somit kann das Modell selbst einschätzen, ob eine Antwort gut oder schlecht war, man nennt diese Vorgehensweise auch das Belohnungssystem. Im dritten Schritt wird eine Methode des maschinellen Lernens angewendet, Reinforcement Learning. Genauer wird im Fall von ChatGPT Reinforcement Learning from Human Feedback (RLHF) benutzt, um die Ausgaben zu verbessern. Je nach Ausgabe des Chats wird das Modell entweder belohnt oder bestraft. Ob eine Antwort nun als schlecht oder gut eingeordnet wird, beurteilt eine Belohnungsfunktion. Dadurch können die Ziele in Zahlen festgelegt und Antworten in sogenannte Ergebniszahlen umgewandelt werden. Je höher das Ergebnis einer Zahl ist, desto wahrscheinlicher ist es, dass eine Antwort belohnt wird. So können personalisierte und sichere Antworten gewährleistet werden.

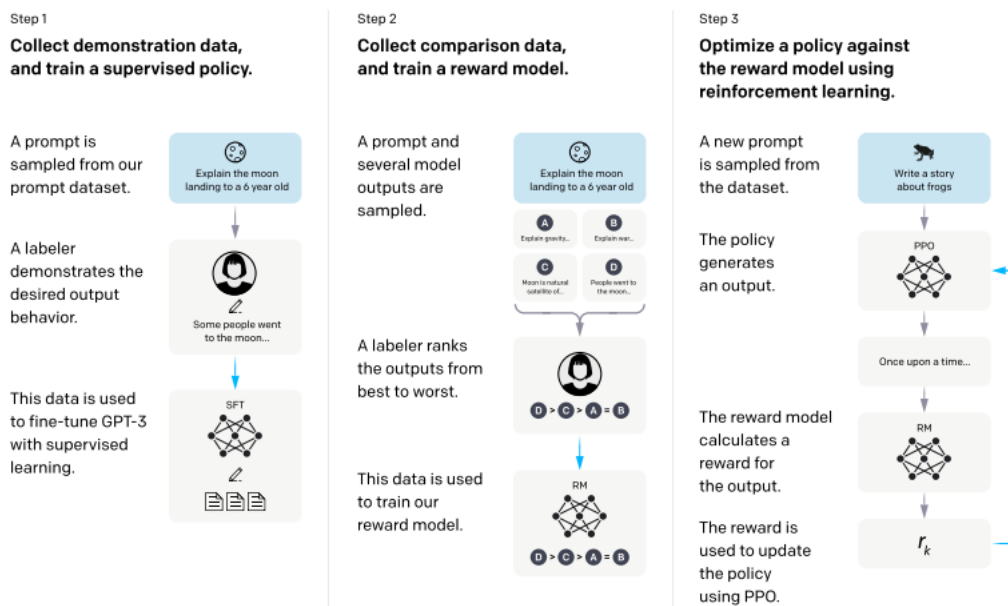


Abbildung 3: ChatGPT's Training
[13]

2.3 Warum ChatGPT

Für die Generierung von Code gibt es mittlerweile viele KI unterstützende Programme. Trotz des umfassenden Angebots habe ich mich für ChatGPT entschieden. Einer der Gründe war der rasante Anstieg an Aufmerksamkeit in den verschiedensten Medien und im persönlichen Umfeld. Ein weiterer wesentlicher Aspekt der mich zur Nutzung von ChatGPT getrieben hat, ist die natürliche Kommunikation mit einem Chatbot. Die Kommunikation fühlt sich sehr intuitiv sowie flüssig an und vor allem menschlich. Gleichzeitig ist man gezwungen, seine Fragen oder Anforderungen präzise zu formulieren, um entsprechende Antworten zu erhalten. Zum menschlichen Faktor zählt auch das ChatGPT eine Konversation halten kann, der Chatbot kann sich innerhalb eines Kontextes auf den bereits vorhandenen Austausch von Nachrichten beziehen und seine Antworten daran anpassen oder verbessern. Auch gibt er zu seinen Antworten Erklärungen und darüber hinaus auch weitere Lösungsvorschläge oder Ideen. Vor allem die Einfachheit des Interagieren mit dem Chatbot hat mich dazu verleitet, ihn dem Copilot von Github³ vorzuziehen, auch wenn dieser auf Codegenerierung spezialisiert ist. Auch ist kein Installieren einer IDE⁴ und der entsprechenden Extensions Voraussetzung um mit ChatGPT zu arbeiten, lediglich ein Webbrowser ist dafür nötig.

2.4 Netline/Crew Optimierer

Der Netline/Crew Optimierer der Lufthansa Systems[18] ist für die optimale Crew Planung in der Luftfahrt zuständig. Die Planung umfasst dabei Crew Mitglieder

³<https://github.com/features/copilot>

⁴Integrated Development Environment

3. Verwandte Arbeiten

wie Piloten, Flugbegleiter und technisches Personal. Die Einsatzpläne berücksichtigen Flugpläne, Regeln und Gesetze, Flotten- und Personalressourcen sowie tarifrechtliche Restriktionen, um optimale Crew Verteilung zu gewährleisten. Durch das effiziente Crew Management können die Kosten einer Airline gesenkt und die Produktivität gesteigert werden. Dies schafft man mit einer optimalen Routenplanung und Auslastung der Crew. Auch werden im Optimierer logistische Probleme einkalkuliert, wie z.B. die Unterbringung der Crew in Hotels, sogenannte Layovers. Der Optimierer ist zu einem Teil in Python geschrieben, worunter auch die zu generierenden Flugregeln und Gesetze gehören.

2.5 Flugregeln und Gesetze

So wie es im Straßenverkehr Gesetze und Regeln gibt, die man befolgen muss, existieren solche auch in der Luftfahrt. Die Regeln werden von der European Union Aviation Safety Agency, kurz EASA⁵, aber auch von den jeweiligen Airlines erstellt. Dabei setzt die EASA Grundlagen fest, die die jeweiligen Airlines zu befolgen haben. Aufgrund der Präferenzen und speziellen Anforderungen jeder Fluggesellschaft, gibt EASA den nötigen Spielraum, um eigene zugeschnittene Regeln zu erstellen. Die Regeln sind dazu da, um die erstellten Crew Zuweisungen auf ihre Korrektheit zu überprüfen. Ein Beispiel für solch eine Regel kann die Mindestruhezeit für die Crew Member nach einem Flug sein. Dabei variiert die Länge der Pause nach Länge der Arbeitsschicht, in der Fliegersprache die sogenannte "Duty". So kann eine Mindestanforderung sein, dass nach einer zehn stündigen Arbeitsschicht eine Ruhezeit von mindestens 24 Stunden benötigt wird. Alle Crew Zuweisungen, die diese Regel missachten, sind somit ungültig und können nicht verwendet werden. Oft unterscheiden sich die Regeln aufgrund der unterschiedlichen Flotte oder regionalen Bedingungen, in denen eine Airline operiert. Die Regeln repräsentieren die einzigartigen Anforderungen und Bedürfnisse der Luftfahrt sowie Airlines wieder, die es im Netline/Crew Optimierer umzusetzen gilt.

3 Verwandte Arbeiten

Es gibt bereits vorhandene Arbeiten, die sich ähnlich mit dem Thema der Codegenerierung auseinandergesetzt haben. In der Arbeit von Yetistiren et al.[24] werden verschiedene Modelle miteinander verglichen und deren Fähigkeiten in Bezug auf Codegenerierung getestet. Dabei werden die erzeugten Code Segmente gegeneinander anhand von verschiedenen Parametern wie Korrektheit und Sicherheit evaluiert. Das Ergebnis der Arbeit verschafft einen Überblick über die jeweiligen Stärken und Schwächen der Modelle. In der Arbeit[24] wurden ChatGPT, Amazon CodeWhisperer und Github Copilot genauer untersucht. Das Ergebnis der Arbeit zeigte, dass ChatGPT von den drei Modellen am besten abgeschnitten hatte, da es die höchste Erfolgsrate bei der Generierung von Code Segmenten erzielte. Ebenfalls wurde erkannt, dass die genaue und präzise Beschreibung eines Problems essentiell für den Erfolg der Generierung war. Eine weitere Arbeit von Tian et al.[19] untersuchte inwiefern ChatGPT

⁵<https://www.easa.europa.eu/en/regulations>

einen Programmierer bei seinen Tätigkeiten unterstützen kann. Die Tätigkeiten beziehen sich dabei auf Codegenerierung, Debugging und Code Zusammenfassung bzw. Erklärung. In der Arbeit wird Code für Leetcode⁶ Aufgaben generiert. Leetcode ist eine Seite, auf der Softwareentwickler algorithmische Probleme lösen können, um ihre eigenen Fähigkeiten zu verbessern. ChatGPT konnte Probleme, die bereits lange existieren, erfolgreich lösen. Aufgaben, die erst nach dem Release von ChatGPT angefertigt wurden, bereiteten dem Modell bemerkbare Schwierigkeiten. Die vor dem Release gestellten Aufgaben könnten Teil der Daten gewesen sein, auf die ChatGPT trainiert worden ist, dies würde den Unterschied erklären. Ebenfalls stellte man fest, je länger die Beschreibungen sind, desto schlechter wurden die Ergebnisse. Daraus schlussfolgerte man den enormen Wert eines qualitativ hochwertig verfassten Prompt und gleichzeitig den Bereich des Prompt Engineerings. Das Potential von LLM's hängt zum Großteil davon ab, wie vertraut der Entwickler mit dem Begriff Prompt Engineering ist. Weitere Arbeiten wie von White et al.[23] und Zhou et al.[25] beschäftigen sich tiefer mit dem Bereich des Prompt Engineerings. White et al.[23] versucht dabei allgemeine Muster zu definieren um den Prozess zur Bestimmung eines effektiven Prompts zu verbessern.

4 Prompt Engineering

Die beeindruckenden Fähigkeiten von Large Language Models hat die Bedeutung sorgfältig ausgewählter Inputs, mit denen sie gefüttert werden, erhöht. Im Fall von ChatGPT kommuniziert man über einen Chat, wie man ihn aus verschiedenen Messengern kennt. Beim Herumspielen mit dem Chatbot war man zu Beginn darüber erstaunt, dass dieser verschiedene Sortieralgorithmen oder einfache Apps programmieren konnte. Nach einer gewissen Zeit, erkannte man das ChatGPT auch darüber hinaus spezifische Probleme lösen konnte, wie z.B. das Schreiben von Dokumentation für den eigenen Code, das Erstellen von Tests oder das Beheben von Bugs und Errors. Damit ChatGPT diese spezifischen Probleme lösen kann, benötigt es sehr präzise und klare Anweisungen. Für den Prozess der Erstellung durchdachter Inputs in Form von natürlicher Sprache gibt es mittlerweile einen Begriff, dieser nennt sich Prompt Engineering[25]. "Prompt engineering is the means by which LLMs are programmed via prompts"[23]. Ein Prompt repräsentiert dabei Befehle, die ein LLM verarbeitet, um einen gewünschten Output zu generieren, wie in unserem Fall die Python Funktionen für die Flugregeln bzw. Gesetze. So gesehen übermittelt ein Prompt dem LLM, welche Informationen wichtig sind und welcher Output erwünscht ist. Somit kann ein Modell darauf gestimmt werden, Code nach einer gewissen Konvention herzustellen, oder Schlüsselwörter in einem Text hervorzuheben und ggf. zu erklären. Die Prompts entscheiden dabei über die Qualität der Outputs.

⁶<https://leetcode.com/>

Anweisung	Eine bestimmte Aufgabe oder Anweisung, die das Modell ausführen soll
Kontext	Ergänzende Details oder Informationen, die das Modell zu verbesserten Antworten führen können
Input Daten	Genaue Informationen oder Daten, die das Modell verarbeiten soll
Output Indikator	Bestimmt Art oder das Format der Ausgabe

Tabelle 1: Die vier Schlüsselemente eines Prompts

4.1 Schlüsselemente eines Prompts

Es gibt mittlerweile verschiedene Techniken, um effektive Prompts zu erstellen. Bevor man überhaupt einen dieser anwenden kann, müssen im Vorfeld einige Schlüsselemente eines Prompts geklärt werden. Insgesamt gibt es vier (Tab.: 1), die einen wirksamen Prompt ausmachen: Anweisung, Kontext, Input Daten und Output Indikator[17]. Die Anweisung beschreibt die Aufgabe und befiehlt dem Modell, was es zu tun hat. Eine Anweisung kann z.B. sein, "fasse einen Text zusammen" oder "erstelle Rezept Ideen". Der Kontext spielt dabei eine wichtige Rolle und sorgt für zusätzliche Informationen für das Modell, um einen besseren Output zu generieren. So könnte man schreiben, "gebe mir Rezeptideen für ein vegetarisches Essen" oder "gebe mir Rezept Ideen, um meine Schwiegereltern zu beeindrucken". Anschließend gibt es noch die Input Daten, die genau definieren, auf welchen Daten der Output generiert werden soll. Im Fall von einer Textzusammenfassung oder einer Erklärung muss ein Absatz oder ein Paragraph als Input mitgeliefert werden, damit ChatGPT diese beim Output berücksichtigen kann. Zusätzlich kann noch ein Output Indikator angegeben werden. Dieser Indikator kann das Format des Outputs festlegen, so kann bestimmt werden, einen besonders kurzen Output zu erstellen, den Output für ein Kleinkind verständlich zu machen oder den Output im Stil eines berühmten Dichters oder ähnlichem zu verfassen. Ebenfalls ist es wichtig zu beachten, dass auch, wenn ChatGPT auf deutsch kommunizieren kann, der Großteil der Daten, auf denen das Modell trainiert worden ist, auf englischen Daten basiert[4]. Somit kann ein besseres Ergebnis erzielt werden, wenn man mit ChatGPT auf englisch kommuniziert. Ebenso gilt je kürzer ein Prompt ist, desto länger kann der Chatbot den Kontext aufrechterhalten und sich auf die bereits übergebenen Daten länger beziehen und im Umkehrschluss bessere Outputs generieren.

4.2 Prompt Techniken

Je nach Erfahrungslevel eines Nutzers können unterschiedliche Prompt Techniken angewandt werden. Ein durchschnittlicher Nutzer wird im Gegensatz zu einer Person mit einem tieferen Verständnis von LLM und Anwendungen wie ChatGPT, nicht von fortschrittlichen Techniken Gebrauch machen. Eher werden Grundtechniken angewendet, um effektive Prompts herzustellen. Einer dieser Grundtechniken ist das Rollenspiel[12]. Bei diesem Vorgehen wird dem Chatbot eine Rolle zugewiesen, beispielsweise die eines Fitnesstrainers, der die Erstellung eines Trainingsplans übernimmt oder die eines Reiseführers, welcher den Ablauf eines Städtetrips in einer bestimmten

Stadt bereitstellt. Durch die Zuteilung einer Rolle kann bereits hier ein besseres Ergebnis erzielt werden, da das Modell in einer spezifischen Ausrichtung gesteuert wurde und somit der wichtige Kontext hergestellt worden ist. Vor allem als Anfänger, aber auch als erfahrener Prompt Engineer, erstellt man nicht immer konstant beim ersten Versuch den perfekten Prompt. Den Erfahrungen zugrunde startet man entweder mit einem mehr oder weniger durchdachten Prompt und übergibt es dem Chatbot. Bei einem unerwünschten Ergebnis kann der Nutzer seinen Prompt nochmals an ausgewählten Stellen verändern und erneut hineingeben. Nach jeder Iteration nähert man sich dem erhofften Ergebnis und kann so zu seinem Ziel kommen. Auf diese Weise entwickelt man mit der Zeit ein natürliches und instinktives Verständnis für diese Anwendung. Man nennt diese Art des Prompten, die Iterative Verfeinerung[9]. Neben den Grundlagen gibt es auch fortgeschrittene Techniken, die ein tieferes Verständnis von ChatGPT benötigen. Die fortgeschrittenen Vorgehensweisen lassen sich dabei in drei grobe Ober Kategorien einteilen: N-Shot Prompting[1], Chain of Thought Prompting[22] und Generated Knowledge Prompting[10]. Beim N-Shot Prompting steht N für die Anzahl an mitgelieferten Beispielen, nämlich wie das Modell eine Aufgabe zu erledigen hat. Dabei gibt es zwei Subkategorien, Zero-Shot Prompting und Few-Shot Prompting. Da ChatGPT auf große Mengen an Daten trainiert ist, kann durchaus mit der Zero-Shot Technik ein gelungenes Ergebnis erzielt werden, wie man anhand des folgenden Beispiels erkennen kann.

Zero-Shot Prompting:

Prompt: [17]
Classify the text into neutral, negative or positive. Text: I think the vacation is okay. Sentiment:
Output:
Neutral

Hier wird eine klare Aufgabe definiert und kein Beispiel oder hilfreicher Kontext geliefert. Beim N-Shot Prompting werden zusätzlich Beispiele bereitgestellt, um die Leistungsfähigkeit zu steigern. Dies kann eine effektivere Nutzung des Modells ermöglichen. Das **Chain of Thought Prompting**[22] erklärt dem Modell die Argumentation seiner Antwort in Schritten. Dadurch imitiert das Modell die Denkweise des Nutzers und formuliert ebenfalls seinen Output in logischen Schritten. Die Technik wird vor allem bei komplexen Problemen angewendet, da die Zerlegung des Problems in Einzelteilen gleichzeitig dem Nutzer hilft, klare Anweisungen zu schreiben und dem Modell einen besseren Output generieren lässt.

4. Prompt Engineering

Chain-of-Thought Prompting:

Prompt: [17]

The odd numbers add up to an even number: 17, 9, 10, 12, 13, 4, 2.

A: Adding all the odd numbers (17, 9, 13) gives 39. The answer is False.

The odd numbers add up to an even number: 15, 32, 5, 13, 82, 7, 1.

A:

Output:

Adding all the odd numbers (15, 5, 13, 7, 1) gives 41. The answer is False.

In die gleiche Kategorie fällt die Self-Consistency Technik[21]. Self-Consistency ist ein Ansatz, der darauf beruht, einfach mehrere Outputs zu generieren und die am häufigsten generierte Antwort als finales Ergebnis anzusehen. Eine weitere fortgeschrittene Technik ist das **Generated Knowledge Prompting**. Im Mittelpunkt dieses Verfahren steht das Wissen, welches der Prompt zusätzlich zur Verfügung gestellt bekommt. Oft ist es üblich, dass man das Modell selbst das Wissen generieren lässt und im nächsten Prompt manuell einfügt.

Generated Knowledge Prompting:

Prompt: [17]

Question: Part of golf is trying to get a higher point total than others. Yes or No?

Knowledge: The objective of golf is to play a set of holes in the least number of strokes. A round of golf typically consists of 18 holes. Each hole is played once in the round on a standard golf course. Each stroke is counted as one point, and the total number of strokes is used to determine the winner of the game.

Explain and Answer:

Output:

No, the objective of golf is not to get a higher point total than others. Rather, the objective is to play a set of holes in the least number of strokes. The total number of strokes is used to determine the winner of the game, not the total number of points.

Das zusätzliche Wissen führt zu einem besseren Ergebnis und Performance des Modells[10]. Egal für welche Technik man sich letztendlich entscheidet, in den häufigsten Fällen wendet man eine Kombination aus verschiedenen Methoden an. Vor allem am Anfang werden die iterativen Prozesse in jedem Fall ein ständiger Begleiter sein, um ein effektives Ergebnis zu erzielen. Ein guter Prompt Engineer zeichnet sich dadurch aus, dass er sich seinen speziellen Fall genauestens bewusst ist und entsprechend die korrekten Techniken verwendet. Dabei sollten die jeweiligen Vorteile einzelner Verfahren analysiert und gegebenenfalls anschließend kombiniert werden,

um zu einem gewünschten Output zu gelangen.

5 Domänenspezifisches Prompting

Im folgenden Abschnitt wird erläutert, wie man Flugregeln und Gesetze in einer bereits vorhandenen Codebase mit eigenen Klassen und Variablen erstellen kann. Dabei muss beim Generieren von domänenspezifischen Code auf viele Details geachtet werden, wie z.B. die korrekte Bezeichnung von Variablen oder deren Datentypen. Zuerst erläutere ich dabei den Prozess, wie ich es geschafft habe, mit der Festlegung der Schlüsselemente und Anwendung verschiedener Techniken einen "base" Prompt zu erstellen, mit dem sich alle Regeln generieren lassen. Mit "base" Prompt meine ich eine Art Grundgerüst, das in jedem Prompt wiederzufinden ist. Aufgrund der Individualität der Regeln unterscheiden sich die Prompts an ausgewählten Stellen, damit jeweils die gewünschte Funktion generiert werden kann. Eine wichtige Stelle dabei ist das Verfassen einer Beschreibung, die alle wichtigen Details einer Regel beinhaltet. So ist jeder Prompt trotz des gemeinsamen Fundaments einzigartig. Anschließend wird das letztendliche Ergebnis, nämlich wie die Prompts aussehen präsentiert. Die darauf folgende Studie untersucht, inwiefern der technische Hintergrund eines Menschen eine Rolle bei der Erstellung von passenden Beschreibungen spielen kann. Gleichzeitig überprüft die Studie die Nutzbarkeit und Verständlichkeit des Skripts. In der Studie gibt es insgesamt fünf Testpersonen, die alle unterschiedliche Erfahrungen im Bereich der Informatik sowie der Anwendung ChatGPT haben. Aus Sicht der Lufthansa Systems ist es von besonderem Interesse herauszufinden, welches Level an Erfahrung eine Person braucht, um Airline spezifische Regeln in Python mit Hilfe von ChatGPT generieren zu können. Somit kann auch ein genaues und differenziertes Urteil darüber gefällt werden, inwiefern ChatGPT domänenspezifischen Code generieren kann und von welchen Faktoren es abhängt.

5.1 ChatGPT Input

Um einen effektiven Prompt zu erstellen, ist zuerst die Feststellung der jeweiligen Schlüsselemente Anweisung, Kontext, Input Daten und Output Indikator für meinen konkreten Fall abzuarbeiten. Anschließend wird eine Kombination aus Prompt Techniken angewendet, um zum gewünschten Endergebnis zu gelangen.

Teil 1 des Prompts:

Anweisung: In der Anweisung teile ich dem Modell mit, welche Aufgabe es zu erfüllen hat. Das Ziel ist es, am Ende die jeweiligen Flugregeln und Gesetze in Python Funktionen generiert zu bekommen. Daher könnte die Anweisung wie folgt lauten: "Generiere mir Python Funktionen". Aufgrund des folgenden Kontextes kann ich auf die konkrete Anweisung, das ChatGPT Python Funktionen erstellen soll, verzichten und so an Wörtern sparen.

Kontext: Das zweite Element des Prompt bestimmt den Kontext, in dem sich das Modell einordnen wird. Dieser Schritt zählt vor allem bei der Generierung von domänenspezifischen Code zu einem der wichtigsten Schritte. Basierend auf die im nächsten Schritt mitgelieferten Python Klassen sollen Funktionen erstellt werden. Dies ist

5. Domänenspezifisches Prompting

essentiell, da dadurch das Modell nicht anfängt, eigene Klassen oder Variablen zu erfinden, die nicht der eigenen Codebase entsprechen. Ohne diese Information könnte man diesen domänenspezifischen Code nicht generieren. Ebenfalls könnte ich sogar genauer werden und sagen, dass er Python Klassen bekommt, mit denen er Airline spezifische Regeln generieren soll. Dagegen spricht die Länge des Prompts. Je kürzer der Prompt ist, desto länger kann der Kontext im Chat aufrechterhalten und somit mehr Regeln generiert werden. Dies spielt vor allem eine Rolle, wenn man voneinander abhängige Regeln generiert. ChatGPT hat nämlich die Fähigkeit, sich auf bereits erstellte Funktionen zu beziehen und diese in anderen wiederzuverwenden. Somit muss die Menge an Informationen immer mit Bedacht gewählt werden, vor allem wenn man den Kontext lange aufrechterhalten möchte.

Input Daten: Der Kontext bezieht sich auf das dritte Element des Prompts, nämlich die Input Daten. Dafür fügen wir dem Prompt die bereits erwähnten Python Klassen Leg und Duty hinzu. Die Klasse Leg definiert die wichtigsten Attribute von einem Flug und Duty, dass es eine Sequenz an Legs ist. Duty ist dabei in der Airline Sprache eine Arbeitsschicht welche aus mehreren Flügen bzw. Legs bestehen kann. Diese Beschreibung übergebe ich aber gar nicht, sondern nur die Klassen. Die Klassen stellen im Endeffekt ein Muster dar, damit sichergestellt wird, dass der generierte Output diesem folgt. So kann ChatGPT auf die jeweiligen Variablen und Datentypen zugreifen, um darauf aufbauend Funktionen zu erstellen. White et al.[23] nennt diese Art des Prompten das Template Pattern. Der Vorteil dieser Methode ist, dass das Ergebnis weiter eingeschränkt wird, damit andere mögliche Generierungen verhindert werden.

Output Indikator: Abgeschlossen wird der Prompt mit einem Output Indikator. Dieser ist vor allem wichtig, wenn man die Ausgabe bzw. das Format des Chatbots genauer kontrollieren möchte. Aufgrund der Tatsache, dass mein Prompt bereits hier schon sehr lang geworden ist, beschloss ich, diesen in zwei Teile aufzuteilen. Somit teile ich hier dem Modell mit, dass er auf den zweiten Teil meines Prompts warten soll. Ohne diesen Indikator fängt das Modell an, zufällige und unerwünschte Outputs zu generieren, mit dem Indikator geschieht das nicht.

Ein weiterer wichtiger Grund für die Aufteilung der Prompts ist die schnelle Erstellung eines neuen "sauberen" Chats. Die Erstellung eines neuen Chats ist besonders in zwei Fällen zu raten. Der erste Fall ist der, dass das Modell nach einer gewissen Länge den Kontext einfach verliert und nicht mehr auf die Klassen und Variablen zugreifen kann. Im zweiten Fall kann es passieren, dass man nach mehreren Versuchen zu keinem gewünschten Output gelangt. Hier empfiehlt es sich ebenfalls dringend einen neuen Chat zu erstellen, damit diese Fehler nicht weiter eingebaut und somit keine Folgefehler erzeugt werden können.

Teil 2 des Prompts:

Nun folgt der zweite Teil des Prompts. In diesem Teil des Prompts werden die konkreten Regeln erstellt. Beim Finden eines allgemein gültigen Prompt fand ich hier relativ schnell heraus, dass aufgrund der Einzigartigkeit der Regeln dies hier nicht so einfach möglich ist. Alle Regeln haben einen spezifischen Typ. Eine Regel kann vom Typ eine Duty oder eine Pairing Regel sein. Dies hat einen Einfluss auf die Zusammenstellung des Kontexts und die Input Daten.

Anweisung: Die Anweisung ist bei beiden Arten der Regeln im Kern identisch, denn es sollen Funktionen generiert werden. Da bereits Python Klassen übergeben

worden sind, muss hier nicht genauer angegeben werden, dass Python Funktionen generiert werden sollen. So kann hier wieder an Wörtern gespart werden.

Kontext: Wie bereits angesprochen, hat der Typ einer Regel Auswirkungen auf die Erstellung des Prompts. Ein weiterer wichtiger Aspekt, der dem Prompt zusätzlich übergeben werden sollte, ist das Argument der Funktion. Die Übergabe des Arguments hilft dem Modell nämlich enorm dabei, die korrekten Attribute und Variablen zu benutzen und die korrekten Operationen durchzuführen. Je nachdem welche Kategorie die Regel letztendlich angehört, ändert sich der Prompt was die Übergabe der Argumente betrifft. Eine Regel, die eine Duty überprüft, bekommt auch als Argument eine Duty. Bei einer Pairing Regel hingegen wird eine Sequenz an Duties als Argument übergeben, dabei ist diese Information beim Generieren solch einer Regel besonders wichtig. Durch das Übergeben der richtigen Argumente wird hier das Modell weiter in die richtige Richtung gesteuert, um ein erfolgreiches Ergebnis zu erzielen.

Input Daten: Um den zweiten Teil des Prompts abzuschließen, benötigt es noch die Beschreibung der Regel, welche die Input Daten darstellen. Da der Inhalt jeder Regel einzigartig ist, muss dies auch im Prompt berücksichtigt werden. Es gilt nun die Beschreibung so zu wählen, dass alle wichtigen Details und Feinheiten der Regel beachtet werden, damit ChatGPT darauf basierend die Funktion erstellen kann. Dies ist eine Stelle im Prompt, die keine feste Vorlage hat, da es aufgrund der Besonderheit der Regeln nicht möglich ist.

Techniken: Bei der Erstellung der Prompts habe ich verschiedene Techniken angewendet. Da man im zweiten Prompt die Regel selbst beschreiben muss, kam es gelegentlich vor, dass die Iterative Verfeinerung[9] benötigt wurde.

Beispiel für iterativen Prompt:

Generate a maxDuty function, the argument is a pairing, which is a sequence of duty. The function checks if the pairing exceeds the maximum amount of duties, which is 4.

Output:

```
def maxDuty(pairing: Sequence[Duty]) -> bool:
    duty_count = 0
    for duty in pairing:
        duty_count += 1
        if duty_count > 4:
            return False # Exceeds the maximum number of duties
                           (4)
    return True # Does not exceed the maximum number of duties
                  (4)
```

5. Domänenspezifisches Prompting

Beispiel für iterativen Prompt:

make it shorter

Output:

```
def maxDuty(pairing: Sequence[Duty]) -> bool:
    return len(pairing) <= 4
```

Im **Beispiel** wurde eine überflüssig lange Python Funktion geschrieben, woraufhin ich ChatGPT bat eine kürzere Funktion zu generieren. Anstatt einen for loop und einen counter zu benutzen, wurde einfach die Länge der Sequenz mit der maximalen Länge eines Pairings, nämlich 4, verglichen. In beiden Teilen des Prompts wurde zum Großteil die Generated Knowledge Prompting[10] Technik angewendet. Im ersten Prompt stellen die zwei Python Klassen das Wissen dar, welches von mir erstellt und nicht generiert worden ist. Mithilfe des Wissens über die eigene Codebase, konnte das Modell die Variablen und Typen für die Generierung der Regeln nutzen. Im zweiten Teil ist die Beschreibung des Wissen, welches dem Modell übergeben wird. Dazu wurde das Zero-Shot Prompting[1] eingesetzt, da ohne weitere Zugabe von bereits geschriebenen Python Funktionen diese auf Anhiob vom Modell angefragt worden sind. Durch die Anwendung verschiedener Techniken und die Aufteilung der Prompts in zwei Teile konnte das Modell auf einen spezifischen Teil der Codebase vorbereitet werden.

1. Teil des Skripts:

Prompt:

I will give you 2 python classes as input. With the classes you will generate functions. Wait for my next instructions.

```
class Leg(Protocol):
    """Abstract description of Leg"""
    depUtc: datetime
    depLt: datetime
    arrUtc: datetime
    arrLt: datetime
    flightDesignator: str
    rotation: Optional[int]
    isTransport: bool = False
    day: int

class Duty(Protocol):
    """Abstract description of Duty"""

    legs: Sequence[Leg]
```

2. Teil des Skripts:

Wenn Duty Regel:

Prompt:

Generate a (insert function name) function, the argument is a duty (or more). The function checks if (insert description) .

Wenn Pairing Regel:

Prompt:

Generate a (insert function name) function, the argument is a pairing, which is a sequence of duty. The function checks if (insert description).

Hier sind die ausführlichen Prompts vorzufinden mit denen es mir gelang alle Regeln zu erstellen. Wenn man nun eine Flugregel generieren möchte muss man die Prompts in der Reihenfolge **Prompt1** und dann **Prompt2** ChatGPT übermitteln. Möchte man in einem Chat mehrere Regeln generieren, muss man den zweiten Prompt entsprechend anpassen und erneut in den Chat eingeben. Der erste Prompt muss nicht wieder dem Modell übergeben werden, erst wenn man wieder einen neuen Chat eröffnen möchte. Die Größe des ersten Prompts ist, wie man erkennen kann, den zwei Klassen Leg und Duty geschuldet. Die anderen Elemente des Prompts sind in den ersten Sätzen wiederzufinden, dabei sind diese in englisch verfasst. Im Gegensatz zum zweiten Prompt muss man hier lediglich den Input kopieren und in den Chat einfügen.

Beim zweiten Prompt muss der Ersteller einer Regel sich zwischen zwei Auswahlmöglichkeiten entscheiden. Dabei muss der Typ der Regel bestimmt werden und entsprechend der Prompt gewählt werden. Die Prompts sind mit Lücken versehen, die von der jeweiligen Person auszufüllen sind. Es muss ein Name für die Funktion und eine passende Beschreibung für die Regel bereitgestellt werden. Falls dies getan ist, kann daraufhin der zweite Prompt ebenfalls übergeben werden.

5.2 Nutzung der Prompts

#	!	Typ	Regel	Beschreibung
1		Duty	Früher Start	Wenn das erste Leg einer Duty vor 0900 LT Abflugzeit beginnt ist es eine Frühschicht
2	1	Pairing	Maximale Frühschichten	Maximal 3 aufeinander folgende Frühschichten erlaubt

Abbildung 4: Zwei Flugregeln

5. Domänenspezifisches Prompting

Im folgenden Abschnitt werde ich zwei beispielhafte Verläufe präsentieren, um zu zeigen, wie mit den Prompts umzugehen ist. Hier in Abb.: 4 sind zwei Flugregeln dargestellt. Einmal eine Regel, die eine Frühschicht bestimmt und eine, die die maximale Anzahl an darauf folgenden Frühschichten kontrolliert. Wie bereits eben erklärt, muss der erste Teil des Prompts ChatGPT übergeben werden, damit dieser auf den folgenden Input vorbereitet ist. Daraufhin fügt man den zweiten Teil des Prompts ein.

Da es sich hier bei der ersten Regel um eine Duty Regel handelt, muss man diesen **Duty Prompt** benutzen. Die Regel kontrolliert genau eine Duty, ob sie früh ist, deswegen bekommt die Funktion auch nur eine als Argument. Zudem nenne ich die Regel "isEarly" da so leicht zu erkennen ist, worum es in der Funktion geht. Zum Schluss füg ich noch eine Beschreibung ein, dabei ist es wichtig alle Details einzubauen. In dieser Regel ist es wichtig zu erwähnen, dass wenn das erste Leg einer Duty vor 9 Uhr morgens lokaler Ortszeit beginnt, es eine Frühschicht ist. Fügt man alle Einzelheiten zusammen, erhält man diesen **Prompt** und den folgenden korrekten Output.

Prompt:

Generate a isEarly function, the argument is a duty. The function checks if the first leg of a duty starts before 0900 LT.

Output:

```
def isEarly(duty: Duty) -> bool:
    """Check if a Duty started before 09:00 LT"""
    first_leg = duty.legs[0] # Erstes Leg der Duty
    return first_leg.depLt.time() < time(9, 0)
```

Bei der zweiten Regel handelt es sich um eine Pairing Regel, dementsprechend muss dieser **Pairing Prompt** ausgewählt werden. Sonst gibt es vom Ablauf der ersten Regel keinen besonderen Unterschied. Beim **Output** der zweiten Regel benutzt das Modell die bereits vorher generierte "isEarly" Funktion. Man erkennt an dieser Stelle auch die Stärke des Kontexts von ChatGPT.

Alle Regeln folgen grob dem gleichen Schema und unterscheiden sich je nach Typ im Fundament und in der Beschreibung. Die Regeln sind auf Github unter folgenden Commit-Hash⁷ zu finden: 7556e31b7133d29442c7d377461e9969959aedd8.

⁷<https://github.com/john-ba23/bachelor-flugregeln/tree/7556e31b7133d29442c7d377461e9969959aedd8>

Prompt:

Generate a maxEarlyDuties function, the argument is a pairing, which is a sequence of duty. The function checks if 3 consecutive early duties occurred.

Output:

```
def maxEarlyDuties(duties: Sequence[Duty]) -> bool:
    early_shift_count = 0 # counter for early duties
    for duty in duties:
        if isEarly(duty):
            early_shift_count += 1
            if early_shift_count > 3:
                return False
        else:
            early_shift_count = 0
    return True
```

5.3 Validierung des Skripts

Das vorherige Kapitel hat bestätigt, dass es im Fall dieser Arbeit möglich ist, domänenspezifischen Code zu generieren. Dabei habe ich für jede Regel die Prompts so angepasst, dass am Ende eine korrekte Python Funktion erstellt worden ist, die den Kern der Regel widerspiegelt. Dafür gab es die offenen Stellen im zweiten Prompt. Im Interesse der Lufthansa Systems gilt es herauszufinden, welches Level an Erfahrung eine Person braucht, um diese Lücken zu füllen. Natürlich mit dem Ziel, dass der generierte Output einer korrekten Python Funktion entspricht. Da mit den Prompts allein eine Person relativ wenig anfangen kann, werden diese in ein Skript eingebaut. Das Skript, welches z.B. ein neuer Mitarbeiter bei der Lufthansa später erhält, fungiert als eine Art Anleitung, um mit den jeweiligen Prompts arbeiten zu können. Für die Studie wird das Skript noch mit einem Fragebogen ergänzt, welches später in der Version für die Lufthansa Systems entfernt wird. Die gestellten Fragen dienen später zur Auswertung, um bei unterschiedlichen Ergebnissen auch gegebenenfalls Rückschlüsse auf die Fähigkeiten und Erfahrungen der jeweiligen Probanden zu ziehen. Somit kann die Lufthansa Systems die Voraussetzung bestimmen, die ein neuer Angestellter haben muss, um diese Regeln erstellen zu können. Zusätzlich könnte man den komplexen Part, nämlich das Verstehen der Codebase und die intensive Auseinandersetzung der Flugregeln überspringen und so die Produktivität innerhalb des Unternehmens steigern und dabei mehr Zeit für den Kern der Anwendung zur Verfügung haben. Die Studie soll zusätzlich die Verständlichkeit und die Effizienz des Skripts validieren und überprüfen, ob es damit überhaupt möglich ist, domänenspezifischen Code zu generieren. Wichtig anzumerken ist, dass es sich hier um eine iterative Studie handelt. Die iterative Studie erlaubt es mir zusätzlich, nach jedem Durchgang mein Skript durch die gesammelten Beobachtungen und anschließender Analyse der Ergebnisse zu verbessern. So besteht die Möglichkeit, bei der nächsten Iteration ein besseres Ergebnis zu erzielen. Das vollständige Skript ist im Anhang [A.1](#) vorzufinden. Ich möchte hier kurz einmal den Aufbau des Skripts und den Ablauf

6. Ergebnisse

der Studie schildern. Bevor die Studie richtig beginnt, werden dem Proband Fragen gestellt um Informationen über den individuellen Erfahrungsstand der Person zu ermitteln. Die Fragen beziehen sich dabei auf die Erfahrung in der Programmierung, die Kenntnisse in der Programmiersprache Python und der Nutzung von ChatGPT. Daraufhin folgt eine Anleitung, die in fünf Schritte aufgeteilt ist. Die Anleitung erklärt unter anderem wie die Testperson mit den Prompts umzugehen hat und was deren konkrete Aufgabe dabei ist. Auf der nächsten Seite sind dann die jeweiligen Prompts vorzufinden und auf der letzten Seite ist nochmal ein Beispiel, welches den Verlauf eines Chats darstellt. Da hier vor allem das Skript untersucht werden soll, habe ich den Probanden vorab mitgeteilt das ich keine inhaltliche Fragen beantworte und sie nur mit Hilfe des Skripts zurecht kommen müssen. Es gab für die Studie kein Zeitlimit, die Person konnte sich mit dem Skript so lange auseinandersetzen wie sie möchte.

6 Ergebnisse

#	1.	2.	3.	4.	5.	6.	7.	8.	9.	Regeln erstellt	Dauer
1	4	J	-	J	Grundlage n	J	2	Anschreiben schreiben, Umformulierung, Code Übersetzung, Konzepte erklären lassen	N	2/4	60min
2	6	J	-	J	viel	J	1	Weiterbildung, lösen domänenspezifischer Fragestellungen (wo ich wenig Vorwissen habe), teilweise als Github Copilot für Autocompletion	J	3/4	64min
3	1,5	N	N	N	keine	J	2	E-Mails, Uniarbeiten	N	2/4	70min
4	7	J	-	J	Grundlage n	J	3	Code review, "rubber duck"	J	4/4	32min
5	0	N	N	N	keine	J	2	Powerpoint Präsentationen, Tabellen erstellen, E-Mails	N	1/4	45min

Abbildung 5: Studienergebnisse (J = Ja, N = Nein)

1. Wie viele Jahre Programmiererfahrung haben Sie? 2. Haben Sie momentan einen Job als Softwareentwickler? 3. Falls nein, haben Sie einen Job im Bereich der Informatik? 4. Haben Sie Erfahrung in der Programmiersprache Python? 5. Falls ja, wie viel Erfahrung haben Sie? 6. Haben Sie Erfahrungen mit ChatGPT? 7. Wie oft benutzen Sie ChatGPT? (0 = nie, 1 = 1x im Monat, 2 = wöchentlich, 3 = täglich) 8. Wofür benutzen Sie ChatGPT? (E-Mail schreiben, Rezepte, Code Generierung) 9. Kennen Sie den Begriff Prompt Engineering?

In Abb.5 kann man in der Tabelle die Beantwortung der Fragen entnehmen und das letztendliche Ergebnis, welches erzielt wurde. Nochmals ist es wichtig zu erwähnen das nicht alle Probanden unter den gleichen Bedingungen die Studie durchgeführt haben. Nach jeder absolvierten Testperson, wurde das Skript mit meinen Beobachtungen und nachträglicher Analyse der Ergebnisse geändert, um in der nächsten Iteration ein besseres Ergebnis zu erzielen. In dieser Studie wurden folgende Regeln (Anhang: A.3) bearbeitet: 1, 3, 4, 10. Die wichtigsten Beobachtung der jeweiligen Probanden werden in diesem Kapitel vorgestellt.

6.1 Studienergebnisse

Testperson 1⁸:

Testperson 1 hat mit einer Programmiererfahrung von vier Jahren und wöchentlicher Nutzung von ChatGPT 2/4 Regeln erstellen können. Die ersten zwei Regeln konnte die Person mithilfe des Skripts automatisch von ChatGPT generieren lassen.

Prompt für Regel 10 von Testperson 1:

Generate a 'maxEarlyDuty' function, the argument is a pairing, which is a sequence of duty. The function checks if at max 3 consecutive early duties occur.

Output:

```
def maxEarlyDuty(pairing: Sequence[Duty]) -> bool:
    """Check if at most 3 consecutive early duties occur in the
       given pairing."""
    consecutive_early_duties = 0
    for duty in pairing:
        if earlyStart(duty):
            consecutive_early_duties += 1
        else:
            consecutive_early_duties = 0
        if consecutive_early_duties > 3:
            return False
    return True
```

Der Proband wirkte selbstsicher und sehr enthusiastisch bei der Generierung der Regeln. Wie man anhand dem Prompt für **Regel 10** erkennen kann, weist der Proband das Modell detailliert daraufhin das maximal drei aufeinander folgende Frühschichten erlaubt sind. Das Modell implementiert einen counter "consecutive_early_duties" welcher False returned, falls mehr als drei Frühschichten in einem Pairing sind. Auch verwendet ChatGPT die bereits in der ersten Regel erstellte earlyStart() Methode wieder.

Bei der **Regel 3** scheiterte es bei der Wahl der Argumente. Das im zweiten Teil des Prompts in Klammer gesetzte "or more", reichte nicht aus, um den Probanden zu verdeutlichen, dass man der Regel auch gezielt eine feste Anzahl an Duties übergeben

⁸<https://chat.openai.com/share/09f75e0f-1c0c-4f5f-8922-9cad69ed6659>

6. Ergebnisse

kann. Dabei hing das Ergebnis enorm von der korrekten Anzahl an Duties ab, die man im Skript selber festlegen muss.

Der Output ist nicht weit von einer korrekten Lösung entfernt. Richtig ist die Berechnung der Pause zwischen zwei Duties. Falsch ist jedoch, dass die Funktion als Argument eine Sequenz von Duties bekommt und so über alle mit einer for Schleife iteriert. Somit kann die Funktion auch nicht für Regel 4 benutzt werden, da dort ein Aufruf mit genau zwei Argumenten benötigt wird. Durch diesen Durchlauf konnte ich mein Skript in der Hinsicht verbessern, dass die verschiedenen Möglichkeiten besser hervorzuheben sind und das Wählen der Argumente eindeutiger gemacht werden muss.

Prompt für Regel 3 von Testperson 1:

Generate a 'reducedBreak' function, the argument is a pairing, which is a sequence of duty. The function checks if the break between two duties is less than 9h and returns True if this is the case.

Output:

```
def reducedBreak(pairing: Sequence[Duty]) -> bool:
    """Check if the break between two duties in the pairing is
    less than 9 hours."""
    for i in range(len(pairing) - 1):
        current_duty = pairing[i]
        next_duty = pairing[i + 1]
        # Calculate the time between the end of the current duty
        # and the start of the next duty
        time_between_duties = next_duty.legs[0].deplT -
                               current_duty.legs[-1].
                               arrLt
        if time_between_duties < timedelta(hours=9):
            return True
    return False
```

Testperson 2⁹:

Mit einer Programmiererfahrung von sechs Jahren und monatlicher Nutzung von ChatGPT konnte Testperson 2 3/4 Regeln erstellen. Dazu konnte diese Person den Begriff Prompt Engineering, welches auch ein Grund für die Verbesserung sein kann. Wie die erste Testperson durchlief auch Testperson 2 die ersten beiden Regeln ohne Probleme. Auch bei Regel 3 konnte die Person durch die korrekte Wahl der Argumente ein korrektes Ergebnis erzielen, welche eine Iteration davor aufgrund des Skripts noch nicht geklappt hat. Es war an Stellen jedoch zu vermerken, dass das Übersetzen der Regel in eine andere Sprache zusätzlich zu einer erhöhten Komplexität beigetragen hat. Regel 4 alleine ist schon recht kompliziert zu verstehen. Diese Regel darauf in eine andere Sprache zu übersetzen, führte in diesem Fall zu einer falschen Beschreibung und letztendlich zu einem fehlerhaften Ergebnis. Der Proband erkannte zwar

⁹<https://chat.openai.com/share/29c38bd9-b577-4820-ab0a-32c2adc0d080>

den Fehler, entschied sich dennoch gegen einen weiteren Prompt, um zu einem richtigen Ergebnis zu gelangen. Der Grund dafür war, dass die Person der Überzeugung war, die Regel selbst schneller korrigieren zu können, anstatt einen neuen Prompt zu schreiben, der diesen Fehler ausbessert. Dies tat diese Person anschließend auch und behob den Fehler, um so zu einem korrekten Ergebnis zu kommen. Trotz der fehlgeschlagenen Regel habe ich bis auf die Hervorhebung der jeweiligen Chat Eingaben mit Hilfe von Farbe nichts Wesentliches verändert.

Testperson 3¹⁰:

Testperson 3 hatte im Gegensatz zu den anderen Teilnehmern mehr Schwierigkeiten beim Generieren der Regeln. Zudem hat er die geringste Programmiererfahrung und keinen Job im Bereich der Informatik. Im Kontrast zu den anderen Probanden, die relativ schnell mit der Erstellung einer passenden Beschreibung begonnen haben, verbrachte die dritte Testperson viel Zeit mit den Einzelheiten der Chat Eingaben. Es kam mir so vor als würde er ChatGPT misstrauen und an dessen Fähigkeiten zweifeln, Code generieren zu können. Ebenfalls verbrachte er viel Zeit damit, das Beispiel zu verstehen. Trotz der investierten Zeit konnte kein klarer Zusammenhang zur Aufgabe geknüpft werden. Hier fiel mir auf, dass zwar ein beispielhafter Ablauf einer erfolgreichen Generierung vorlag, jedoch die dazugehörige Regel, wie sie letztendlich im Regel Dokument steht, fehlte. Dies fügte ich bei der nächsten Iteration hinzu. Erneut ist aufgefallen, dass die übergebene Beschreibung bei Regel 10 nicht korrekt ist. Dies kann entweder damit zu tun haben, dass die Regel nicht richtig verstanden worden ist, oder dass beim Übersetzen in die englische Sprache der Kern der Regel verloren gegangen ist. Meine Vermutung ist, dass es eine Kombination aus beiden ist. Wie ich bereits erwähnt hatte, kam es mir so vor als würde der Proband die Fähigkeiten des Modells misstrauen und sich deswegen für eine recht komplizierte, aber nicht unbedingt detaillierte Beschreibung entscheiden. Das Ergebnis waren komplex geformte Sätze, die nicht mehr der Original Erklärung entsprachen. So konnte dann auch keine korrekte Regel generiert werden. Durchaus kann die fehlende Programmiererfahrung dazu geführt haben, sich eher aufs Verstehen des Codes zu fokussieren anstatt auf die konkrete Aufgabe. Aufgrund der erneuten Beobachtung, dass die Übersetzung eine mögliche zusätzliche Hürde bei der Erstellung einer Beschreibung ist, wird die nächste Person als Unterstützung ein Übersetzungstool ihrer Wahl zur Verfügung gestellt bekommen, falls dies benötigt wird.

Testperson 4¹¹:

Die vierte Testperson hat mit einer Programmiererfahrung von sieben Jahren und täglicher Nutzung von ChatGPT alle Regeln erfolgreich generiert. Ebenfalls war die Testperson mit 32 Minuten Bearbeitungszeit die schnellste von allen Probanden. Selbst die letzte Regel konnte die Person erfolgreich generieren, auch wenn er dies nicht mit nur einem Prompt geschafft hat. Im Vergleich zu den anderen lieferte diese Testperson auch ein Beispiel mit, wie die letzte Regel abstrakt auszusehen hat, wie es im Dokument auch zu sehen ist (Anhang A.3). Die Testperson erkannte die Fehler in der letzten generierten Funktion und musste mit zwei weiteren Prompts nachhelfen. Bei der letzten Eingabe wies der Proband den Chatbot auf eine ungenutzte Variable hin, woraufhin er diese entfernte. So kam es auch bei der letzten Regel zu einem kor-

¹⁰<https://chat.openai.com/share/c3c378b6-a56c-48df-bd3d-519efbaf3213>

¹¹<https://chat.openai.com/share/b4c8b303-cb12-4a78-be5d-beddc75b7854>

rekten Ergebnis. Ich konnte während der Durchführung feststellen, dass die Person sehr selbstbewusst und effizient an die Aufgabe herangeht. Gründe dafür können die vorhandenen Erfahrung in der Softwareentwicklung aber auch ChatGPT sein. Zudem benutzt er ChatGPT auch für Codespezifische Aufgaben.

Testperson 5¹²:

Die letzte Testperson hat keine Programmierkenntnisse oder sonstiges Wissen über die Informatik. Aus Sicht der Lufthansa Systems, aber auch der eigenen, ist es interessant herauszufinden, wie viele Regeln jemand ohne dieses Wissen erstellen kann. Wichtig zu erwähnen ist, dass die Person wöchentlich ChatGPT benutzt und im Umgang mit dieser Anwendung zumindest vertraut ist. Da die folgende Testperson keinerlei Erfahrungen im Bereich der Informatik besitzt, habe ich vorab das Skript daran angepasst. Vor allem bei der Auswahl des zweiten Teil des Skripts, änderte ich die Formulierung. Ich nehme an, dass sich die Testperson nichts unter dem Begriff "Argument" vorstellen kann. So lenkte ich den Fokus bei der Bestimmung der Auswahlmöglichkeiten auf die Wahl zwischen einer Duty oder Pairing Regel. Die Testperson muss "nur" feststellen, ob es sich um eine Pairing oder Duty Regel handelt. Bei einer Duty Regel muss die Person noch zusätzlich unterscheiden, um wie viele Duties es sich handelt und entsprechend angeben. Es wurde 1/4 Regeln erfolgreich generiert. Während der Erstellung des Prompts konnte ich feststellen, dass im Vergleich zu den anderen Probanden wichtige Details in der Beschreibung zum Teil ausgelassen worden sind.

Prompt von Testperson 5 für Regel 1:

Generate a an early start function, the argument is a duty. The function checks if if the leg starts before 9am it will be an early shift

Output:

```
def mark_early_shift(duty: Duty) -> None:
    """Mark legs with an early start before 9 AM as an early
        shift."""
    early_start_time = datetime.strptime("09:00", "%H:%M").time()
    ()

    for leg in duty.legs:
        if leg.depLt.time() < early_start_time:
            leg.isEarlyShift = True
        else:
            leg.isEarlyShift = False
```

Wie im Beispiel von **Testperson 5** zu sehen ist, wurde nicht konkret angegeben, dass das erste Leg einer Duty kontrolliert werden soll. Stattdessen wurde sehr ungenau der Begriff Leg benutzt, was dazu führte, dass er über alle Legs einer Duty iterierte. Dies war aber nicht gefordert. Bei der nächsten Regel tritt ein ähnlicher Fehler auf.

¹²<https://chat.openai.com/share/367a5a7c-d352-49c8-b557-1c06c9c2f265>

Prompt von Testperson 5 für Regel 10:

Generate a Maximum early shift sequence function, the argument is a pairing, which is a sequence of duty. The function checks if the maximum shifts are applied.

Hier wird das wichtige Element vergessen, drei aufeinander folgende Frühschichten zu kontrollieren. Jedoch wird einfach nur geschrieben das ChatGPT eine maximale Schicht Anzahl erlauben soll. Weder die Anzahl der Schichten, noch dass es sich dabei um Frühschichten handelt, wurde übergeben. Trotz falscher Beschreibungen konnte die Testperson die verschiedenen Auswahlmöglichkeiten im 2. Teil des Skripts richtig den Regeln zuordnen. Ein interessante Erkenntnis ist, dass für Personen mit Informatik Hintergrund, die Wichtigkeit von Details und vor allem Zahlen nicht außer Acht gelassen werden. Denn vor allem die Feinheiten machen am Ende den Erfolg einer korrekten Generierung aus.

Das Ergebnis repräsentativ für diese Gruppe an Menschen zu interpretieren, ist nicht eindeutig festzustellen. Der Grund dafür ist dass nur eine Testperson mit dieser Voraussetzung an dieser Studie teilgenommen hat. Auch wenn nur eine von vier Regeln generiert worden ist, wurden die Regeln den korrekten Wahlmöglichkeiten zugeordnet. Letztendlich scheiterte es an der eigenen Beschreibung der Regel, die sehr von der korrekten Übergabe der Feinheiten abhängt. Eine weiterer Hinweis an den Probanden, die Regeln detailliert zu beschreiben, oder eine kleine Einführung könnte das Skript in diesem Punkt verbessern.

In dieser Studie wurde nun untersucht wie Menschen mit unterschiedlichen Voraussetzungen mit dem Skript und vor allem mit den Prompts umgehen. Insgesamt zeigte die Studie, dass ohne weitere Hilfestellungen als mit dem Skript es möglich ist, domänenspezifischen Code zu generieren. Die Qualität der Ergebnisse hing zum einen vom Skript, aber auch von der individuellen Erfahrung der Probanden ab. In Abb.5 kann man erkennen, dass die Probanden mit den meisten Erfahrungen in der Programmierung auch die meisten Regeln erstellen konnten. Auch gibt es nicht eine Person in der Studie, die es nicht geschafft hat mindestens eine Regel zu generieren. Ein möglicher Grund dafür ist, dass alle Probanden die Prompts den entsprechenden Regeln korrekt zuordnen konnten. So konnte auch die letzte Testperson ohne jeglichen technischen Hintergrund eine Regel erstellen. Die Studie hebt besonders hervor, wie wichtig das Verfassen einer genauen und präzisen Beschreibung ist, da davon oft der Erfolg des Outputs abhing. Ebenfalls konnte festgestellt werden, dass die Komplexität der Aufgabe mit dem Übersetzen der Regel in die englische Sprache erhöht wurde. Der offensichtliche Nachteil einer Person ohne Informatik Hintergrund ist die Tatsache, dass sie nicht in der Lage sein wird, die generierte Regel überprüfen zu können. Wohingegen eine Person mit dem Wissen den Output kontrollieren und gegebenenfalls eine weiteren Prompt schreiben könnte. Eine mögliche Option, um trotzdem diese Gruppe von Menschen einzusetzen, wäre die Erstellung eines Test Frameworks. Das Framework müsste über eine Schnittstelle Zugriff auf die generierten Regeln bekommen und diese anschließend automatisch testen. So könnte die Person die Regel auf ihre Korrektheit kontrollieren und gegebenenfalls einen neuen Prompt schreiben. Ohne den Prompts und dem darum gebauten Skript würde der Prozess der Gene-

rierung des Code anders ablaufen. Ein neuer Angestellter müsste sich erst mit der Codebase auseinandersetzen und grundlegende Kenntnisse in der Programmiersprache Python mitbringen, um diese dann zu verstehen. Dies kann viel Zeit beanspruchen und hängt vor allem von der Erfahrung des jeweiligen Entwicklers ab. Ebenfalls müsste sich der Entwickler intensiv mit dem Regel Dokument auseinandersetzen, um anschließend Python Funktionen schreiben. Dabei müssen die Funktionen den Kern der Regel widerspiegeln. Der Prozess wird um einiges länger dauern als sich kurz mit dem Skript und den jeweiligen Prompts auseinanderzusetzen. Zudem muss sich hier zumindest am Anfang nicht intensiv mit der Codebase auseinandergesetzt werden.

6.2 Skript für Lufthansa Systems

Die gewonnenen Erkenntnisse aus der Studie führten zu einem iterativ verbesserten Skript (Anhang A.4) für die Lufthansa Systems. Letztendlich unterscheidet sich das Skript nicht großartig zur letzten Version (Anhang A.2), außer dass z.B. der Fragebogen entfernt und die Anleitung angepasst worden ist. Mit einer kleinen mündlichen Einführung in das Regelwerk und einem besonderen Verweis auf die Wichtigkeit der Details, sehe ich noch bessere Erfolgchancen auf ein gelungenes Ergebnis als bereits in der Studie erzielt worden ist. Eine weitere Verbesserungsmöglichkeit wäre das Regel Dokument selbst. Die Beschreibungen könnten bereits mit dem Hintergedanken an ChatGPT formuliert werden. Das Resultat wäre eine effizientere und vor allem effektivere Generierung der Regeln. Gepaart mit einem automatischen Test Framework könnte so ein Großteil der Arbeit erledigt werden. Dies führt zwar zu einem erheblichen Zeitaufwand, wenn man aber die Menge an Regeln bedenkt, wäre dies eine potentielle Option.

7 Fazit

In dieser Arbeit wurde untersucht, inwiefern ChatGPT domänenspezifischen Code generieren kann. Mit der Aneignung verschiedener Prompt Techniken und dem Bestimmen der Schlüsselemente eines Erfolg versprechenden Prompts, wurde ein "base" Prompt zusammengestellt. Mit dem konnten alle Regeln generiert werden. Wichtig anzumerken ist, dass bei der Konstruktion der Prompts stets auf die Anzahl an Wörtern geachtet wurde. Anfangs bestand nämlich die Befürchtung, dass die Übergabe von notwendigen Informationen das Limit des Kontexts sprengen könnte. Im "base" Prompt waren Lücken eingebaut, um Platz für die spezifischen Anforderungen der jeweiligen Regeln übrig zu lassen. So mussten die Prompts mit einem Namen für die Regel und einer Beschreibung versehen werden. Dabei war die Erstellung einer präzisen und genauen Beschreibung der entscheidende Faktor, ob das Modell korrekten Code generiert. Die in der Arbeit durchgeführte Studie untersuchte, inwiefern die individuellen technischen Erfahrungen mit dem Ergebnis des Modells zusammenhängen. Es haben dabei fünf Testpersonen teilgenommen, die alle einen verschiedenen Grad an Erfahrungen hatten. Je höher die Erfahrungen waren, desto besser waren die erzielten Ergebnisse. Eine Testperson konnte alle Regeln in der Studie absolvieren. Die unterschiedlichen Ergebnisse sind auf die jeweiligen Beschreibungen zurückzuführen.

Dabei sind die Beschreibungen vom jeweiligen Verfasser abhängig. Ein ausgeprägter technischer Hintergrund sorgte eher für die Erstellung präziser und akkurater Beschreibungen, als wenn dieser geringer vorhanden ist. Auch hob die Studie hervor, dass mit den Prompts sehr effizient Code erstellt werden kann, Testperson 4 hat für alle Regeln 32 Minuten gebraucht. Durch die vermehrte Anerkennung der Fähigkeiten von Large Language Modellen, wie ChatGPT, wird der Begriff Prompt Engineering in Zukunft zunehmend an Bedeutung gewinnen. Mit einem fundierten Verständnis, wie das Modell funktioniert, und einem Überblick über das Prompt Engineering, kann man ChatGPT sehr effizient nutzen. Durch die Aufteilung der Prompts und das Vermeiden unnötiger Informationen konnte im konkreten Fall das Problem des limitierten Kontexts umgangen werden. Solange also die nötigen Informationen nicht die Grenzen des Kontexts überschreiten, kann mit Hilfe der gezeigten Methoden und Techniken domänenspezifischer Code generiert werden. Somit kann ChatGPT einen Entwickler bei der Erstellung von effizientem und korrekten Code unterstützen.

Literatur

- [1] Tom Brown u. a. "Language models are few-shot learners". In: *Advances in neural information processing systems* 33 (2020), S. 1877–1901.
- [2] Peter Buxmann und Holger Schmidt. "Grundlagen der künstlichen Intelligenz und des maschinellen Lernens". In: *Künstliche Intelligenz: Mit Algorithmen zum wirtschaftlichen Erfolg* (2021), S. 3–25.
- [3] KR1442 Chowdhary und KR Chowdhary. "Natural language processing". In: *Fundamentals of artificial intelligence* (2020), S. 603–649.
- [4] the decoder. *ChatGPT Guide: Prompt-Strategien für bessere Ergebnisse*. URL: <https://the-decoder.de/chatgpt-guide-prompt-strategien/>.
- [5] Christian Janiesch, Patrick Zschech und Kai Heinrich. "Machine learning and deep learning". In: *Electronic Markets* 31.3 (2021), S. 685–695.
- [6] Vijay Kakani u. a. "A critical review on computer vision and artificial intelligence in food industry". In: *Journal of Agriculture and Food Research* 2 (2020), S. 100033.
- [7] Joost N Kok u. a. "Artificial intelligence: definition, trends, techniques, and cases". In: *Artificial intelligence* 1 (2009), S. 270–299.
- [8] Yann LeCun, Yoshua Bengio und Geoffrey Hinton. "Deep learning". In: *nature* 521.7553 (2015), S. 436–444.
- [9] Zhexin Liang u. a. "Iterative prompt learning for unsupervised backlit image enhancement". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2023, S. 8094–8103.
- [10] Jiacheng Liu u. a. "Generated knowledge prompting for commonsense reasoning". In: *arXiv preprint arXiv:2110.08387* (2021).
- [11] Batta Mahesh. "Machine learning algorithms-a review". In: *International Journal of Science and Research (IJSR).[Internet]* 9.1 (2020), S. 381–386.
- [12] Rakesh K Maurya. "Using AI Based Chatbot ChatGPT for Practicing Counseling Skills through Role-play". In: (2023).
- [13] Long Ouyang u. a. "Training language models to follow instructions with human feedback". In: *Advances in Neural Information Processing Systems* 35 (2022), S. 27730–27744.
- [14] Scalable path. *How Does ChatGPT Actually Work? An ML Engineer Explains*. URL: <https://www.scalablepath.com/data-science/chatgpt-architecture-explained>.
- [15] Alexander Maximilian Röser. *Charakterisierung von schwacher und starker Künstlicher Intelligenz*. 79. Arbeitspapiere der FOM, 2021.
- [16] Sofia Samoilu u. a. "AI Watch. Defining Artificial Intelligence. Towards an operational definition and taxonomy of artificial intelligence". In: (2020).
- [17] Elvis Saravia. "Prompt Engineering Guide". In: <https://github.com/dair-ai/Prompt-Engineering-Guide> (Dez. 2022).

- [18] Lufthansa Systems. *NetLine/Crew*. URL: <https://www.lhsystems.de/solutions/ground-operations-solutions/netlinecrew/>.
- [19] Haoye Tian u. a. "Is ChatGPT the Ultimate Programming Assistant—How far is it?" In: *arXiv preprint arXiv:2304.11938* (2023).
- [20] Ashish Vaswani u. a. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).
- [21] Xuezhi Wang u. a. "Self-consistency improves chain of thought reasoning in language models". In: *arXiv preprint arXiv:2203.11171* (2022).
- [22] Jason Wei u. a. "Chain-of-thought prompting elicits reasoning in large language models". In: *Advances in Neural Information Processing Systems* 35 (2022), S. 24824–24837.
- [23] Jules White u. a. "A prompt pattern catalog to enhance prompt engineering with chatgpt". In: *arXiv preprint arXiv:2302.11382* (2023).
- [24] Burak Yetiştiren u. a. "Evaluating the Code Quality of AI-Assisted Code Generation Tools: An Empirical Study on GitHub Copilot, Amazon CodeWhisperer, and ChatGPT". In: *arXiv preprint arXiv:2304.10778* (2023).
- [25] Yongchao Zhou u. a. "Large language models are human-level prompt engineers". In: *arXiv preprint arXiv:2211.01910* (2022).

A. Anhang

A Anhang

A.1 Erste Version des Skripts

Sehr geehrter Teilnehmer/-in,

in der folgenden Studie möchte ich Sie bitten, mit Hilfe von ChatGPT Python Funktionen zu erstellen. Vorab möchte ich Ihnen noch ein paar Fragen stellen.

1. Wie viele Jahre Programmiererfahrung haben Sie? _____
 2. Haben Sie momentan einen Job als Softwareentwickler? _____
 3. Falls nein, haben Sie einen Job im Bereich der Informatik? _____
 4. Haben Sie Erfahrung in der Programmiersprache Python? _____
 5. Falls ja, wie viel Erfahrung haben Sie? _____
 6. Haben Sie Erfahrungen mit ChatGPT? _____
 7. Wie oft benutzen Sie ChatGPT? _____ (0 = nie, 1 = 1x im Monat, 2 = wöchentlich, 3 = täglich)
 8. Wofür benutzen Sie ChatGPT? (E-Mail schreiben, Rezepte, Code Generierung)
-
9. Kennen Sie den Begriff Prompt Engineering? _____

Sie werden Fluggesetze und Regeln für den Netline/Crew Optimierer von Lufthansa Systems generieren. Die Regeln sind Teil einer riesigen Codebase, in der Studie gilt es zu überprüfen, ob es möglich ist, diese Regeln ohne das Domänenspezifische Wissen zu programmieren. Dafür wird ihnen ein Skript zur Verfügung gestellt, die Studie wird folgend ablaufen:

1. Machen Sie sich mit den Flugregeln 1, 3, 4 und 10 vertraut
2. Schauen Sie sich das Beispiel an
3. Öffnen Sie chat.openai.com
4. Fügen Sie Teil 1 des Skriptes in den Chat ein
5. Fügen Sie Teil 2 des Skriptes mit der von Ihnen gewählten Beschreibung in den Chat ein
6. Wenn Sie mit Ihrem Ergebnis zufrieden sind, können Sie Schritt 5. mit der nächsten Regel wiederholen

Skript:

Erste Eingabe:

I will give you 2 python classes as input. With the classes you will generate functions. Wait for my next instructions.

```
class Leg(Protocol):
    """Abstract description of Leg"""

    depUtc: datetime
    depLt: datetime
    arrUtc: datetime
    arrLt: datetime
    flightDesignator: str
    rotation: Optional[int]
    isTransport: bool = False
    day: int

class Duty(Protocol):
    """Abstract description of Duty"""

    legs: Sequence[Leg]
```

Zweite Eingabe:

Wenn Duty Regel:

Generate a _____ function, the argument is a duty (or more). The function checks if _____ .

Wenn Pairing Regel:

Generate a _____ function, the argument is a pairing, which is a sequence of duty. The function checks if _____ .

Beispiel:

JO

I will give you 2 python classes as input. With the classes you will generate functions. Wait for my next instructions.

```
class Leg(Protocol):
    """Abstract description of Leg"""

    depUtc: datetime
    depLt: datetime
    arrUtc: datetime
    arrLt: datetime
    flightDesignator: str
    rotation: Optional[int]
    isTransport: bool = False
    day: int

class Duty(Protocol):
    """Abstract description of Duty"""

    legs: Sequence[Leg]
```



Certainly, I have the two Python classes you provided: `Leg` and `Duty`. Please go ahead with your instructions on generating functions based on these classes, and I'll assist you accordingly.



JO

Generate an `maxDuty` function, the argument is a pairing, which is a sequence of duty. The function checks if the pairing exceeds the maximum amount of duties, which is 4.



To generate a `maxDuty` function that checks if a pairing (sequence of duties) exceeds the maximum amount of duties (4 in this case), you can create a Python function like this:



python

✂ Save to grepper

📄 Copy code

```
from typing import Sequence

def maxDuty(pairing: Sequence[Duty]) -> bool:
    """Check if the pairing exceeds the maximum amount of
    return len(pairing) > 4
```

In this function, we import the `Sequence` type hint to specify that `pairing` is expected to be a sequence of `Duty` objects. Then, we simply check if the length of the `pairing` sequence is greater than 4 and return `True` if it is, indicating that the maximum number of duties has been exceeded, or `False` otherwise.

A.2 Letzte Version des Skripts

Sehr geehrter Teilnehmer/-in,

in der folgenden Studie möchte ich Sie bitten, mit Hilfe von ChatGPT Python Funktionen zu erstellen. Vorab möchte ich Ihnen noch ein paar Fragen stellen.

1. Wie viele Jahre Programmiererfahrung haben Sie? _____
 2. Haben Sie momentan einen Job als Softwareentwickler? _____
 3. Falls nein, haben Sie einen Job im Bereich der Informatik? _____
 4. Haben Sie Erfahrung in der Programmiersprache Python? _____
 5. Falls ja, wie viel Erfahrung haben Sie? _____
 6. Haben Sie Erfahrungen mit ChatGPT? _____
 7. Wie oft benutzen Sie ChatGPT? _____ (0 = nie, 1 = 1x im Monat, 2 = wöchentlich, 3 = täglich)
 8. Wofür benutzen Sie ChatGPT? (E-Mail schreiben, Rezepte, Code Generierung)
-
9. Kennen Sie den Begriff Prompt Engineering? _____

Sie werden Fluggesetze und Regeln für den Netline/Crew Optimierer von Lufthansa Systems generieren. In der Studie gilt es zu überprüfen, ob es möglich ist, diese Regeln ohne das Domänenspezifische Wissen zu programmieren. Dafür wird Ihnen ein Skript zur Verfügung gestellt, die Studie wird folgend ablaufen:

1. Machen Sie sich mit den Flugregeln 1, 3, 4 und 10 vertraut
2. Schauen Sie sich das Beispiel auf Seite 3 an, hier wird einmal der Vorgang mit einer Beispiel Regel visuell dargestellt. Von Ihnen wird nicht verlangt, die Einzelheiten des Skripts zu verstehen oder des Programmcodes, es dient lediglich dazu, wie ein typischer Verlauf aussehen soll.
3. Öffnen Sie chat.openai.com
4. Konzentrieren Sie sich bei der Erstellung der Regel auf eine effektive Beschreibung der Regel. Die Beschreibung soll in einer natürlichen Sprache erfolgen.
5. Fügen Sie Teil 1 des Skripts in den Chat ein (rot markiert) und warten Sie auf den Output von ChatGPT.
6. Fügen Sie Teil 2 vom Skript (blau markiert) mit einer passenden Beschreibung der Regel in den Platzhaltern (schwarz markiert) ein, verfassen Sie die bitte in englisch. Fangen Sie mit der Regel 1 an und beachten Sie dabei auf die Hinweise der zweiten Seite.
7. Wenn Sie mit Ihrem Ergebnis zufrieden sind, können Sie Schritt 5. mit der nächsten Regel wiederholen

Skript:

Erste Eingabe:

I will give you 2 python classes as input. With the classes you will generate functions. Wait for my next instructions.

```
class Leg(Protocol):
    """Abstract description of Leg"""
```

```
    depUtc: datetime
    depLt: datetime
    arrUtc: datetime
    arrLt: datetime
    flightDesignator: str
    rotation: Optional[int]
    isTransport: bool = False
    day: int
```

```
class Duty(Protocol):
    """Abstract description of Duty"""
```

```
    legs: Sequence[Leg]
```

Zweite Eingabe:

Entscheiden Sie sich bitte für einen der drei Möglichkeiten, je nachdem ob es eine Duty oder Pairing Regel ist.

Hinweis: Achten Sie darauf, dass bei Duty Regeln auch eine bestimmte Anzahl an Duties gefordert sein kann.

Regel mit genau einer Duty:

Generate a (insert name for function here) function, the argument is a duty.
The function checks if (insert description here).

Regel mit mindestens zwei Duties:

Generate a (insert name for function here) function, the arguments are (1,2,3,...) duty.
The function checks if (insert description here).

Regel mit einem Pairing:

Generate a (insert name for function here) function, the argument is a pairing, which is a sequence of duty. The function checks if (insert description here).

Beispiel:

#	!	Typ	Regel	Beschreibung
1		Pairing	Maximale Pairing Länge	Die Maximale Pairing Länge beträgt 4 Tage

JO

I will give you 2 python classes as input. With the classes you will generate functions. Wait for my next instructions.

```
class Leg(Protocol):
    """Abstract description of Leg"""

    depUtc: datetime
    depLt: datetime
    arrUtc: datetime
    arrLt: datetime
    flightDesignator: str
    rotation: Optional[int]
    isTransport: bool = False
    day: int

class Duty(Protocol):
    """Abstract description of Duty"""

    legs: Sequence[Leg]
```



Certainly, I have the two Python classes you provided: `Leg` and `Duty`. Please go ahead with your instructions on generating functions based on these classes, and I'll assist you accordingly.



JO

Generate an `maxDuty` function, the argument is a pairing, which is a sequence of duty. The function checks if the pairing exceeds the maximum amount of duties, which is 4.



To generate a `maxDuty` function that checks if a pairing (sequence of duties) exceeds the maximum amount of duties (4 in this case), you can create a Python function like this:



python

✂ Save to grepper

📄 Copy code

```
from typing import Sequence

def maxDuty(pairing: Sequence[Duty]) -> bool:
    """Check if the pairing exceeds the maximum amount of
    return len(pairing) > 4
```

In this function, we import the `Sequence` type hint to specify that `pairing` is expected to be a sequence of `Duty` objects. Then, we simply check if the length of the `pairing` sequence is greater than 4 and return `True` if it is, indicating that the maximum number of duties has been exceeded, or `False` otherwise.

A.3 Regeldokument



Flugregeln für Bachelorarbeit

#	!	Typ	Regel	Beschreibung
1		Duty	Früher Start	Wenn das erste Leg einer Duty vor 0900 LT Abflugzeit beginnt ist es eine Frühschicht
2		Duty	Späte Ankunft	Wenn das letzte Leg einer Duty nach 2100 LT endet ist es eine Spätschicht
3		Rest	Reduzierte Pause	Wenn die Pause zwischen zwei Duties geringer als 9h ist, ist es eine reduzierte Pause
4	1, 3	Pairing	Kein Früher Start nach reduzierter Pause	Wenn eine Duty aufhört und die darauffolgende Pause eine reduzierte Pause ist, dann darf die übernächste Duty kein Früher Start sein Duty -> reduzierte Pause -> Duty -> Pause -> no Early Duty
5	1,2	Pairing	Kein Früher Start nach später Ankunft	Wenn eine Duty nach 2100 LT Ankommt dann darf die draufkommende duty kein Früher Start sein, außer die Duty beginnt mit einem Tag Pause dazwischen
6	3	Pairing	Maximale Anzahl an Flügen nach reduzierter Pause	Duty -> reduzierte Pause -> Duty mit maximal 3 Legs und einem Transport (Deadhead)
7		Cico	Check in und Check out	Check in 1h vor Flug, Check out 30min nach Flug
8		Pairing	Deadhead und Flug	Minimale Zeit zwischen Deadhead und Flug beträgt 1h40 wenn ein Deadhead das erste Leg eines Pairings ist, sonst 50min
9		Rest	Zeit zwischen Flugzeugwechsel	Minmale Zeit zwischen Flugzeugwechsel: 50min
10	1	Pairing	Maximale Frühschichten	Maximal 3 aufeinander folgende Frühschichten erlaubt
11		Pairing	Maximale Anzahl an Duties	Maximale Anzahl an Duties in einem Pairing ist 4
12		Pairing	Maximale Pairing Länge	Die Maximale Pairing Länge beträgt 4 Tage
13		Pairing	Maximale Transporte in Pairing	Maximale Transporte pro Pairing ist maximal 10 Stück
14		Pairing	Maximale Flugzeug wechsel	Maximale Anzahl an Flugzeug Wechsel innerhalb eines Pairings
15		Duty	Duty überlappt 2 Tage	Eine Duty darf sich nicht über zwei Tage erstrecken

A.4 Skript für Lufthansa Systems

Sie werden Flugregeln und Gesetze für den Netline/Crew Optimierer von Lufthansa Systems generieren. Dafür wird Ihnen ein Skript zur Verfügung gestellt, die Erstellung der Regeln wird folgend ablaufen:

1. Öffnen Sie chat.openai.com
2. Konzentrieren Sie sich bei der Erstellung der Regel auf eine effektive Beschreibung der Regel und achten Sie dabei auf Details. Die Beschreibung soll in einer natürlichen Sprache erfolgen.
3. Fügen Sie Teil 1 des Skripts in den Chat ein (rot markiert) und warten Sie auf den Output von ChatGPT.
4. Fügen Sie Teil 2 vom Skript (blau markiert) mit einer passenden Beschreibung der Regel in den Platzhaltern (grün markiert) ein, verfassen Sie die bitte in englisch. Fangen Sie mit der Regel 1 an und beachten Sie dabei auf die Hinweise der zweiten Seite.
5. Wenn Sie mit Ihrem Ergebnis zufrieden sind, können Sie Schritt 4. mit der nächsten Regel wiederholen

Hinweis:

Schauen Sie sich das Beispiel auf Seite 3 an, hier wird einmal der Vorgang mit einer Beispiel Regel visuell dargestellt. Von Ihnen wird nicht verlangt, die Einzelheiten des Skripts zu verstehen oder des Programmcodes, es dient lediglich dazu, wie ein typischer Verlauf aussehen soll.

Skript:

Erste Eingabe:

I will give you 2 python classes as input. With the classes you will generate functions. Wait for my next instructions.

```
class Leg(Protocol):
    """Abstract description of Leg"""

    depUtc: datetime
    depLt: datetime
    arrUtc: datetime
    arrLt: datetime
    flightDesignator: str
    rotation: Optional[int]
    isTransport: bool = False
    day: int

class Duty(Protocol):
    """Abstract description of Duty"""

    legs: Sequence[Leg]
```

Zweite Eingabe:

Entscheiden Sie sich bitte für einen der drei Möglichkeiten, je nachdem ob es eine Duty oder Pairing Regel ist.

Hinweis: Achten Sie darauf, dass bei Duty Regeln auch eine bestimmte Anzahl an Duties gefordert sein kann.

Regel mit genau einer Duty:

Generate a (insert name for function here) function, the argument is a duty.
The function checks if (insert description here).

Regel mit mindestens zwei Duties:

Generate a (insert name for function here) function, the arguments are (1,2,3,..) duty.
The function checks if (insert description here).

Regel mit einem Pairing:

Generate a (insert name for function here) function, the argument is a pairing, which is a sequence of duty. The function checks if (insert description here).

Beispiel:

#	!	Typ	Regel	Beschreibung
1		Pairing	Maximale Pairing Länge	Die Maximale Pairing Länge beträgt 4 Tage

JO

I will give you 2 python classes as input. With the classes you will generate functions. Wait for my next instructions.

```
class Leg(Protocol):
    """Abstract description of Leg"""

    depUtc: datetime
    depLt: datetime
    arrUtc: datetime
    arrLt: datetime
    flightDesignator: str
    rotation: Optional[int]
    isTransport: bool = False
    day: int

class Duty(Protocol):
    """Abstract description of Duty"""

    legs: Sequence[Leg]
```



Certainly, I have the two Python classes you provided: ``Leg`` and ``Duty``. Please go ahead with your instructions on generating functions based on these classes, and I'll assist you accordingly.



JO

Generate an `maxDuty` function, the argument is a pairing, which is a sequence of duty. The function checks if the pairing exceeds the maximum amount of duties, which is 4.



To generate a ``maxDuty`` function that checks if a pairing (sequence of duties) exceeds the maximum amount of duties (4 in this case), you can create a Python function like this:



python

✂ Save to grepper

📄 Copy code

```
from typing import Sequence

def maxDuty(pairing: Sequence[Duty]) -> bool:
    """Check if the pairing exceeds the maximum amount of
    return len(pairing) > 4
```

In this function, we import the ``Sequence`` type hint to specify that ``pairing`` is expected to be a sequence of ``Duty`` objects. Then, we simply check if the length of the ``pairing`` sequence is greater than 4 and return ``True`` if it is, indicating that the maximum number of duties has been exceeded, or ``False`` otherwise.